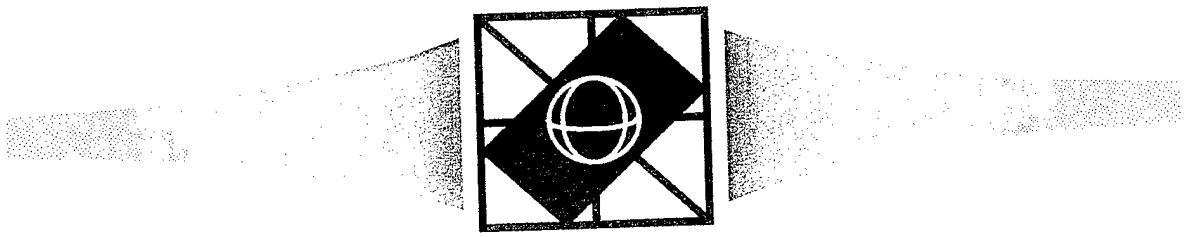


Thirteenth Annual IFIP Working Group 11.3 Conference on Database Security

The Roosevelt Hotel, Seattle, Washington
July 25 – 28, 1999

CONFERENCE PROCEEDINGS

Conference Organizers

Program Co-Chairs:	Vijay Atluri, Rutgers University, U.S.A.
	John Hale, University of Tulsa, U.S.A.
General Chair:	Sujeet Shenoi, University of Tulsa, U.S.A.
IFIP WG11.3 Chair:	John Dobson, University of Newcastle, U.K.
Conference Sponsor:	Office of Naval Research, U.S.A.

Thirteenth Annual IFIP WG 11.3 Working Conference on Database Security
The Roosevelt Hotel, Seattle, Washington, USA, July 25-28, 1999

July 25, 1999, Sunday
14:00 - 19:00 Arrival and Check-in

19:30 - 21:00 Dinner

July 26, 1999, Monday
7:30 - 9:00 Breakfast

9:00 - 9:30 Introductory Remarks

9:30 - 10:30 *SESSION 1: KEYNOTE: Howard Schmidt, Director of Information Security, Microsoft Corporation*
Chair: John Hale - University of Tulsa

10:30 - 11:00 Break

11:00 - 12:30 *SESSION 2: Intrusion Detection* (Chair: T.C. Ting - University of Connecticut)

Intrusion Confinement by Isolation in Information Systems
Peng Liu, Sushil Jajodia and Catherine McCollum - George Mason University/MITRE

Analyzing the Performance of Program Behavior Profiling for Intrusion Detection
Anup K. Ghosh and Aaron Schwartzbard - Reliable Software Technologies

Integrating Data Mining Techniques with Intrusion Detection
Ravi Mukkamala, Jason Gagnon and Sushil Jajodia - Old Dominion University/George Mason University

12:30 - 14:00 Lunch

14:00 - 15:30 *SESSION 3: Role-Based Access Control* (Chair: Sylvia Osborn - University of Western Ontario)

RBAC on the Web by Secure Cookies
Joon S. Park, Ravi Sandhu and SreeLatha Ghanta - George Mason University

eMEDAC: Role-Based Access Control Supporting Discretionary and Mandatory Features
I. Mavridis, G. Pangalos and M. Khair - Thessaloniki University, Greece/Notre Dame Univ., Lebanon

Agent Approaches to Enforce Role-Based Security in Distributed and Web-Based Computing
S.A. Demurjian, Y. He, T.C. Ting and M. Saba - University of Connecticut

15:30 - 16:00 Break

16:00 - 17:00 *SESSION 4: PANEL: Key Issues in Critical Infrastructure Protection*
Participants: Terry Mayfield - Institute for Defense Analyses (Moderator)
Donald Marks - NIST; William Maconachy - DoD; Thomas Harper - PNNL

17:00 - 18:30 Business Meeting

July 27, 1999, Tuesday
7:30 - 9:00 Breakfast

9:00 - 10:00 *SESSION 5: Policy/Modeling* (Chair: John Dobson - University of Newcastle, U.K.)

A Secret Splitting Method for Assuring the Confidentiality of Electronic Records
Andrew Po-Jung Ho - UCLA

For Unknown Secrecies Refusal is Better than Lying
Joachim Biskup - University of Dortmund, Germany

10:00 - 11:00 *SESSION 6: INVITED LECTURE: Cyber Control and Command Research*
Speaker: Catherine McCollum - DARPA
Chair: John Hale – University of Tulsa

July 27, 1999, Tuesday (contd.)

11:00 - 11:30 Break

11:30 - 12:30 *SESSION 7: Workflow Systems* (Chair: Reind van de Riet - Free University, The Netherlands)

Extending the BFA Workflow Authorization Model to Express Weighted Voting
Savith Kandala and Ravi Sandhu - CygnaCom Solutions/George Mason University

A Strategy for an MLS Workflow Management System
Myong Kang, Judith Froscher, Brian Eppinger and Ira Moskowitz - Naval Research Laboratory

12:30 - 14:00 Lunch

14:00 - 15:30 *SESSION 8: Data Mining/Data Warehousing* (Chair: Bhavani Thuraisingham - MITRE)

Impact of Decision-Region Based Classification Mining Algorithms on Database Security
Tom Johnsten and Vijay Raghavan - University of Southwestern Louisiana

Protecting Against Data Mining through Samples
Chris Clifton - MITRE

Security Administration for Federations, Warehouses and Other Derived Data
Arnon Rosenthal, Edward Sciore and Vinti Doshi - MITRE

15:30 - 16:00 Break

16:00 - 17:00 *SESSION 9: Panel: Intrusion Detection*
Participants: T.Y. Lin - San Jose State University (Moderator)
Bhavani Thuraisingham - MITRE; Shayne Pitcock & M.Y. Huang - Boeing; T.C. Ting - Univ. of Connecticut

19:30 - 21:00 Dinner

July 28, 1999, Wednesday
7:30 - 9:00 Breakfast

9:00 - 10:00 *SESSION 10: Multilevel Security* (Chair: Pierangela Samarati - University of Milan)

Enforcing Integrity While Maintaining Secrecy
Donald Marks - NIST

The Effect of Confidentiality on the Structure of Databases
Adrian Spalka and Armin Cremers - University of Bonn, Germany

10:00 - 11:00 *SESSION 11: Temporal Authorization Models* (Chair: Ravi Sandhu - George Mason Univ.)

Temporal Authorization in the Simplified Event Calculus
Steve Barker - University of Westminster, U.K.

Specifying and Computing Hierarchies of Temporal Authorizations
E. Ferrari, P.A. Bonatti, E. Ferrari and M.L. Sapino - University of Milan

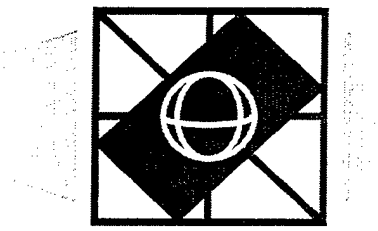
11:00 - 11:30 Break

11:30 - 12:30 *SESSION 12: Object-Oriented Databases* (Chair: Martin Olivier – Rand Afrikaans University)

The Security Problem against Inference Attacks on Object-Oriented Databases
Yasunori Ishihara, Toshiyuki Morita and Minoru Ito - Nara Institute/Hitachi, Japan

A Logical Formalization for Specifying Authorizations in Object-Oriented Databases
Yun Bai and Vijay Varadharajan - University of Western Sydney, Australia

12:30 - 14:00 Lunch



Thirteenth Annual IFIP Working Group 11.3 Conference on Database Security

Conference Organizers

Program Co-Chairs:	**Vijay Atluri, Rutgers University, U.S.A.**
	John Hale, University of Tulsa, U.S.A.
General Chair:	**Sujeet Shenoi, University of Tulsa, U.S.A.**
IFIP WG11.3 Chair:	**John Dobson, University of Newcastle, U.K.**
Conference Sponsor:	**Office of Naval Research, U.S.A.**

TABLE OF CONTENTS

TABLE OF CONTENTS
(Contd.)

Intrusion Detection Session

Monday, July 26, 1999

Chair: T.C. Ting
University of Connecticut

Intrusion Confinement by Isolation in Information Systems *

Peng Liu[2] Sushil Jajodia[1,2] Catherine D. McCollum[1]

[1]The MITRE Corporation
McLean, VA 22102-3481
{jajodia, mccollum}@mitre.org

[2]Center for Secure Information Systems
and
Department of Information and Software Engineering
George Mason University
Fairfax, VA 22030-4444
{pliu, jajodia}@isse.gmu.edu

Abstract

System protection mechanisms such as access controls can be fooled by authorized but malicious users, masqueraders, and misfeasors. Intrusion detection techniques are therefore used to supplement them. However, damage could have occurred before an intrusion is detected. In many computing systems the requirement for a high degree of soundness of intrusion reporting can yield poor performance in detecting intrusions, and can cause long detection latency. As a result, serious damage can be caused either because many intrusions are never detected or because the average detection latency is too long. The process of bounding the damage caused by intrusions during the process of intrusion detection is referred to as intrusion confinement. *We justify the necessity for intrusion confinement during detection by a probabilistic analysis model, and propose a general solution to achieve intrusion confinement. The crux of the solution is to isolate likely suspicious actions before a definite determination of intrusion is reported. We also present a concrete isolation protocol in the file system context to evaluate the feasibility of the general solution, which can be applied in many types of information systems.*

*Jajodia and McCollum were partially supported by Rome Laboratory, Air Force Material Command, USAF, under agreement number F30602-97-1-0139.

1 Introduction

Recently there has been increasing emphasis on supplementing protection of networks and information systems with intrusion detection [Lun93, MHL94, LM98] and numerous intrusion detection products have emerged commercially. Recognizing that access controls, filtering, and other protection mechanisms can be defeated or bypassed by would-be attackers who take advantage of remaining vulnerabilities, intrusion detection systems monitor system or network activity to discover attempts to disrupt or gain illicit access to systems. Intrusion detection must be concerned both with attempts by external penetrators to enter or interfere with the system and by authorized users to exceed their legitimate access or abuse the system in some way. The latter case also includes *seemingly* authorized users, such as masqueraders operating under another user's identification (id) and password, or outside attackers who successfully gained system access but eluded detection of the method of entry. The methodology of intrusion detection can be roughly classed as being either based on *statistical profiles* or on known patterns of attacks, called *signatures.*

Statistical profile-based system compare relevant data by statistical or other methods to representative profiles of normal, expected activity on the system or network. Deviations indicate suspicious behavior. In these systems, there are stringent requirements on not only reporting an intrusion accurately (this is necessary because abnormal behavior is not always an intrusion) but also detecting as many

intrusions as possible (usually, not all intrusions can be detected). However, these two requirements can often result in conflicting design goals.

To make this tangible, consider a system where short-term behavior of a user U_i is audited and compared to U_i's profile, which represents U_i's normal long-term behavior. Whenever U_i's short-term behavior is sufficiently unlike U_i's long-term behavior, a warning is raised. The deviation of U_i's short-term behavior from his long-term behavior can be quantified with the distance, denoted d, in an N-dimensional space from the point defined by a vector of intrusion-detection measures that represents U_i's short-term behavior, to the point defined by a vector that represents U_i's long-term behavior. Now the problem is: How long a d is sufficient?

Based on the assumption that the more significant the deviation, the larger the possibility that U_i's short-term behavior is an intrusion, in order to ensure a high degree of soundness of intrusion reporting (the soundness can be measured by the probability that when a short-term behavior is reported as an intrusion it is really an intrusion), d needs to be made sufficiently long. However, as d gets longer, the number of intrusions that can be detected decreases because intrusions characterized by gradual anomaly can be overlooked by the detector (a formal analysis is presented in Section 2), in which case many intruders may stay at large and cause substantial damage. Moreover, even for an intrusion which is characterized by significant anomaly, choosing a very long d may still cause a long latency of the detection. As a result, substantial damage can be caused by an intruder within the latency. Choosing a shorter d can mitigate these problems; however, the soundness of intrusion reporting can be dramatically degraded. In other words, innocent users may be mistaken for malicious ones and legal service requests may be denied in many situations. This is because trustworthy users may gradually change their behavior in a non-significant way.

With the requirement of a given degree of soundness, the process of bounding the damage caused by undetected intrusions and/or detected intrusions during their latencies is referred to as *intrusion confinement*. Intrusion confinement also encompasses corresponding strategies and mechanisms taken to enable the process.

Signature-based detection examines sniffer logs, audit data, or other data sources for evidence of operations, sequences, or techniques known to be used in particular types of attacks. Signature-based detection techniques cannot be used to detect new, unanticipated patterns that could be detected by statistical profile-based detection techniques. However, they are used to detect known attacks. Intrusion confinement can also be used in association with signature-based detection. We can view a signature as a sequence of actions, as well as the state transitions associated with these actions. Since a prefix of a signature is usually not a signature (otherwise the signature can be shorter), by the time a signature is matched serious damage may have already been caused. Since in many situations such damage may not be recoverable, it would be desirable if we could avoid such damage without causing denial of service.

The first contribution of this paper is that it gives a simple probabilistic model to measure the effectiveness of an intrusion detection system and to justify the necessity of intrusion confinement. Based on the degree of effectiveness, the system security officer (SSO) can decide whether to enforce intrusion confinement and, if so, what mechanisms and strategies to apply for intrusion confinement.

The second contribution of the paper is that it gives a general solution of intrusion confinement which is closely coupled with intrusion detection mechanisms and can be applied in many kinds of information systems. The basic idea is to isolate suspicious actions before an intrusion is reported. However, the decisions about when to isolate and which actions to isolate are made by exploiting the functionality of intrusion detection facilities. Since suspicious behavior may turn out to be an intrusion and cause damage, such isolation can bound the potential damage.

When a suspicious behavior is discovered, the intrusion detector or the SSO must decide how to react and whether to allow continued access by the associated subject, say B, such as a process, a user, or a host. The system can let B continue to access the system just as systems do today, risking further damage, or take action to deny B continued access to the system, which may also be undesirable, for a couple of reasons. Further investigation may show that the suspicious behavior was actually unusual but legitimate activity. If this is the case, denying B access could result in unwarranted denial of service. It is even possible that an attacker is intentionally spoofing B to provoke a denial of service response against B. On the other hand, if B proves guilty, immediately denying access to the system may mean losing the opportunity to gather more information that would help identify the attacker and the objectives of the attack. Experience with system and network penetrations has shown the usefulness of another alternative of intrusion confinement: *isolation*.

Isolating B transparently into a separate environment which still appears to B to be the actual system allows B's activities to be kept under surveillance without risking further harm to the system. An isolation strategy that has been used in such instances is known as *fishbowling*. Fishbowling involves setting up a separate lookalike host or file system and transparently redirecting the suspicious entity's re-

quests to it. This approach allows the incident to be further studied to determine the real source, nature, and goal of the activity, but has some limitations, particularly when considered at the application level. First, the substitute host or file system is essentially sacrificed during the suspected attack to monitor B, consuming significant resources that may be scarce. Second, since B is cut off from the real system, if B proves innocent, denial of service could still be a problem. While some types of service B receives from the substitute, fishbowl system may be adequate, in other cases the lack of interaction with the real system's resources may prevent B from continuing to produce valid results. On the other hand, if the semantics of the application are such that B can continue producing valid work, this work will be lost when the incident concludes even if B is deemed innocent and reconnected to the real system. The fishbowling mechanism makes no provision for re-merging updates from the substitute, fishbowl system back into the real system.

Within the general solution, we offer an isolation strategy that, while in some sense a counterpart to fishbowling, takes advantage of action semantics to avoid some of these limitations. In this isolation approach, as in the case of fishbowling, when B comes under suspicion, we let B continue working while we attempt to determine whether there is anything to worry about. At the same time, we isolate the system from any further damage B might have in mind. However, we provide this isolation without consuming duplicate resources to construct an entirely separate environment, we allow options for partial interaction across the boundary, and we provide algorithms for smoothly merging B's work back into the real system should B prove innocent.

We illustrate our approach in the context of a file system. Consider the following attack scenario: Suppose a user B of the file system comes under suspicion. To let B continue working and at the same time to isolate the file system from any further damage from B, we propose to create an on-the-fly "separate file store" for the user B. All accesses of B are then directed only to the separate suspicious file store, while actions from other users are applied to the main file store. The two stores may become inconsistent over time. Any inconsistency can be resolved when a determination is reached about whether the suspicious user B is actually malevolent or is innocent after all. If B turns out to be malicious, the suspicious file store can be discarded to protect the main store from harm. If B turns out to be innocent, B's updates in the suspicious store will be merged back into the main store; conflicting updates will be identified so that conflicts can then be resolved. The benefit of this scheme is that work can continue while the investigation is in progress. Disruption of the system's operations is kept to a minimum if B is innocent, but B would have been prevented from doing further damage to the system if B is guilty.

This work has relevance to information warfare (IW) defense [AJMB97, GSM96, MG96a, MG96b, PG99]. As pointed out by Ammann, et al. [AJMB97], information warfare defense does everything possible to prevent attacks from succeeding, but it also recognizes that attempting to prevent information attack is insufficient; attacks that are successful to some degree must be recognized as unavoidable, and comprehensive support for identifying and responding to attacks is required.

The rest of the paper is organized as follows. In Sections 2 and 3, we use a probabilistic model to justify the necessity of intrusion confinement. Section 4 presents a general solution of intrusion confinement. In Section 5, we evaluate the feasibility of our solution by presenting a concrete isolation protocol which is applied to a file system. Section 6 discusses related work. In Section 7, we conclude the paper.

2 Why Is Intrusion Confinement Necessary?

Informally, suspicious behavior is the behavior that may have already caused some damage, or may cause some damage later on, but was not reported as an intrusion when it happened. In our model, suspicious behavior emerges in four situations:

1. In statistical profile-based detection, as discussed above, in order to get a high degree of soundness of intrusion reporting, some intrusions characterized by gradual deviations may stay undetected. The corresponding behaviors can be reported as suspicious.

2. In statistical profile-based detection, for a detection with a long latency, the corresponding behavior can be reported as suspicious in the middle of the latency.

3. In statistical profile-based detection, legitimate behavior can be reported as suspicious if it is sufficiently unlike the corresponding profile.

4. In signature-based detection, partial matching of a signature can trigger a report of suspicious behavior.

In the remainder of this section, we present a probabilistic model for justifying the necessity of intrusion confinement. Note that all of the statistics used in this section were computed based on the audit trail, where the entire intrusion history is assumed to be recorded.

2.1 Intrusion Confinement for Statistical Profile-based Detection

Consider an statistical profile-based detection system where a user U_i accesses the system through *sessions*. A session of U_i begins when U_i logs in and ends when U_i logs out. A *behavior* of U_i is a sequence of actions that can last across the boundaries of sessions. A short-term behavior of U_i is a behavior that is composed of a sequence of U_i's most recent actions. The length of the sequence, which is usually small, is determined by the SSO. In contrast, a long-term behavior of U_i is also a sequence of U_i's most recent actions but it is much longer than the short-term behavior. We assume the intrusion detector is triggered in every m actions (or m audit records), that is, after m new actions are executed, both the current short-term behavior and long-term behavior of U_i will be upgraded and the deviation of the new short-term behavior from the new long-term behavior will be computed. When a short-term behavior is upgraded, its oldest m actions will be discarded and the newest m actions will be added.

In the following presentation, we use "behavior" to denote a short-term behavior for brevity. The access history of U_i can be specified as a sequence of behaviors, although two behaviors in the sequence can have multiple actions in common. We model the access history of U_i as a discrete stochastic process $x(t)$ where $x(t_i)$, which indicates the type of the behavior of U_i at time t_i, can take one of three possible values, namely, `LEGITIMATE`, `SUSPICIOUS` and `INTRUSION`, usually with different probabilities. It should be noted that $x(t)$ is theoretically not an independent process because: (1) the behavior of U_i at time t_{i-1} is a part of the long-term behavior that is the profile of the behavior of U_i at time t_i, hence $x(t)$ is affected by $x(t-1)$; (2) the behavior of U_i at times t_{i-1} and t_i can share multiple actions; (3) the behavior of U_i at times t_{i-1} and t_i can have semantic relationships. For example, a `LATEX` command is usually followed by a `DVIPS` command when a `TEX` file is compiled. However, it is reasonable to simplify the model by viewing $x(t)$ as an independent process in many situations when the profile covers a relatively long long-term behavior and when the overlap of neighbored behaviors is only a small part of each behavior because at this point the effect of the behavior of U_i at time t_{i-1} on the profile is too weak to affect $x(t)$, and neighbored behaviors can be viewed as having no overlaps. For simplicity, we model $x(t)$ in the following presentation as an independent discrete stochastic process. In addition, we assume that the access histories of any two users U_i and U_j are independent of each other. Note that our goal is showing that intrusion confinement is necessary, as opposed to evaluating accurately the performance of intrusion detection systems.

We assume that statistical methods, such as the methods proposed in NIDES [JV94], are used in the system. We assume that both the short-term and long-term behaviors of U_i at time t are described by a vector of n intrusion detection measures (or variables). We denote them as $\vec{v_s} = (v_{s1}, ..., v_{sn})$ and $\vec{v_l} = (v_{l1}, ..., v_{ln})$, respectively. The deviation of $\vec{v_s}$ from $\vec{v_l}$ is specified by the distance, denoted $d(\vec{v_s}, \vec{v_l})$, from the point defined by $\vec{v_s}$ to the point defined by $\vec{v_l}$ in the n-dimension space. We further assume that the longer $d(\vec{v_s}, \vec{v_l})$ is, the bigger the probability that $\vec{v_s}$ is an intrusion. In the system, a warning is raised if $d(\vec{v_s}, \vec{v_l})$ is sufficiently long. Interested readers can refer to [JV94] for such details as to which measures can be used, how these measures are quantified, and how $d(\vec{v_s}, \vec{v_l})$ is computed.

We evaluate the effectiveness of such an statistical profile-based detection system with two measures:

- The *rate of detection*, denoted R_d, is the percentage of the intrusions that can be detected among all of the intrusions. R_d indicates the general ability of a detector in detecting intrusions.
- The *rate of errors*, denoted R_e, is the probability that a reported intrusion is actually not an intrusion. R_e indicates the soundness of a detector. The smaller R_e is, the higher degree of soundness is achieved.

R_d and R_e can be formalized based on the parameters listed in Figure 1 as follows. We introduce D_s, P_s, and A_s to model intrusion confinement. Note that D_s is less than D_i; hence, a reported intrusion must be suspicious. The basic meaning of these parameters is shown in Figure 2.

$$R_d = \frac{A_i P_i}{A_s P_s + (1 - A_s) P_g}$$
$$R_e = 1 - P_i$$

Example 1 *Assume the SSO decides that P_i should be at least 0.90 to ensure a high degree of soundness. As a result, the corresponding D_i can be determined to ensure such a P_i. However, ensuring such a high P_i does not mean we can also detect most of the intrusions. If we consider the situation where $A_i = 0.01$, $A_s = 0.10$, $P_s = 0.50$, and $P_g = 0.002$, then the rate of detection $R_d = 0.174$, which is very low. On the other hand, if we decrease the value of P_i to detect more intrusions, for example, we set the value of D_i to that of D_s, then the rate of error $R_e = 1 - P_s = 0.50$, which is too high.*

The above example shows that in many situations if we want to achieve a low rate of errors, then we cannot achieve a high rate of detection. Therefore, the two requirements

D_i	the value of distance such that if $d(\vec{v_s}, \vec{v_l}) \geq D_i$, then $\vec{v_s}$ is reported as an intrusion.
P_i	the probability that when a behavior is reported as an intrusion, that is, $d(\vec{v_s}, \vec{v_l}) \geq D_i$, it is really an intrusion.
D_s	the value of distance such that if $d(\vec{v_s}, \vec{v_l}) \geq D_s$, then $\vec{v_s}$ is reported as suspicious.
P_s	the probability that when a behavior is reported as suspicious, that is, $d(\vec{v_s}, \vec{v_l}) \geq D_s$, it is an intrusion.
A_i	the probability that a behavior deviates from the corresponding long-term behavior with $d(\vec{v_s}, \vec{v_l}) \geq D_i$.
A_s	the probability that a behavior deviates from the corresponding long-term behavior with $d(\vec{v_s}, \vec{v_l}) \geq D_s$.
P_g	the probability that a behavior with $d(\vec{v_s}, \vec{v_l}) \leq D_s$ is an intrusion.

Figure 1. Evaluation Parameters for an Statistical Profile-based Detection System

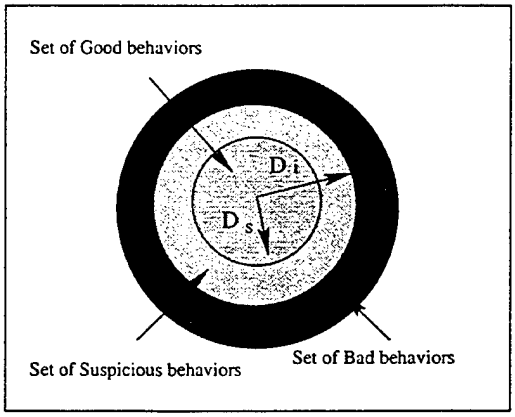


Figure 2. Classification of User Behaviors

can often result in conflicting design goals. Since the rate of errors cannot be very high because otherwise substantial legitimate service requests will be denied, a high rate of detection cannot be achieved in many cases. Thus, many intrusions can stay undetected (in Example 1, the undetected intrusion rate is 82.6%), and the latency of an intrusion can be very long. As a result, serious damage can be caused.

Intrusion confinement can bound this damage. As shown in Figure 1, the set of actions that should be isolated can be specified as follows. Note that since D_s is determined by the SSO based on the value of P_s he prefers, intrusion confinement systems can be flexibly configured.

Definition 1 A short-term behavior described by $\vec{v_s}$ is *suspicious* if $d(\vec{v_s}, \vec{v_l}) \geq D_s$. Here, D_s is determined by the value of P_s, which is chosen by the SSO.

By isolating suspicious behaviors, we can often protect the system from the damage caused by most of the intrusions. The effectiveness of isolation can be measured by the *rate of isolation*, denoted R_i, which is the percentage of the intrusions that will be isolated among all of the intrusions. R_i is formalized as follows:

$$R_i = \frac{A_s P_s}{A_s P_s + (1 - A_s) P_g}$$

In Example 1, if we keep P_i at 0.90, R_i is 96.5%.

2.2 Intrusion Confinement for Signature-based Detection

We specify a signature as a sequence of events leading from an initial limited access state to a final compromised state [PK92, Ilg93, IKP95, SG91, SG97, LWJ98]. Each event causes a state transition from one state to another state. We identify a signature with length n, denoted $Sig(n)$, as $Sig(n) = s_0 E_1 s_1 ... E_n s_n$, where E_i is an event and s_i is a state, and E_i causes the state transition from s_{i-1} to s_i. For simplicity, intra-event conditions are not explicitly shown in $Sig(n)$, although they are usually part of a signature.

A *partial matching* of a signature $Sig(n)$ is a sequence of events that matches a prefix of $Sig(n)$. According to previous discussion, a partial matching is usually not an intrusion. However, it can predict that an intrusion specified by $Sig(n)$ may occur. The accuracy of the prediction of a partial matching, denoted $s_0 E_1 s_1 ... E_m s_m$, can be measured by the following parameter:

P_m : the probability that the partial matching can lead to an intrusion later. Assume the number of the behaviors that match the prefix is N_p and the number of the intrusions that match the prefix is N_i, then $P_m = N_i / N_p$.

Intrusion confinement in signature-based detection is necessary because when an intrusion is reported, serious damage may have already been caused by the intrusion. In signature-based detection, the set of actions that should be isolated is defined as follows. Isolating suspicious behavior can surely confine damage in signature-based detection because the behavior that is actually an intrusion will, with a high probability, be prevented from doing harm to the system.

Definition 2 In signature-based detection, a behavior is *suspicious* if it matches the prefix of a signature but not the whole signature, and P_m of the prefix is greater than or equal to a threshold that is determined by the SSO.

3 When Should Intrusion Confinement Be Enforced?

The decision of whether to enforce intrusion confinement depends on the amount of damage that can be caused. The more serious the damage is, the more efforts the SSO would like to take. In signature-based detection, the decision of whether to enforce intrusion confinement on a known attack that is specified by a signature is dependent on the seriousness of the damage that will be caused by the attack and the value of P_m for each prefix of the signature. For example, if the damage is strongly undesirable, and there exists a prefix whose P_m is sufficiently large, then intrusion confinement can be enforced by isolating the behavior that matches the prefix.

In statistical profile-based detection, however, it can be tricky to make the decision. As shown in Section 2, since degrading the requirement on R_e (the rate of errors) usually can improve R_d (the rate of detection), the SSO may want to find a trade-off between R_e and R_d; thus, the cost of isolation would be avoided. However, a satisfactory trade-off may not be achievable in some systems since the relationship between these two effectiveness measures can dramatically differ from one system to another.

Consider two systems with the same set of parameters and the same associated values as in Example 1. When P_i is degraded from 0.90 to 0.85 in both of the systems, the other parameters may take the values that are listed in the following table. It is easy to get the following result: In system 1, $R_d = 0.82$ and $R_e = 0.15$; In system 2, $R_d = 0.33$ and $R_e = 0.15$. It is clear that when P_i is degraded to 0.85, system 1 performs much better than system 2. At a result, the SSO may need only to enforce intrusion confinement in system 2.

System No.	P_i	P_s	A_i	A_s	P_g
1	0.85	0.50	0.05	0.10	0.002
2	0.85	0.50	0.02	0.10	0.002

It should be noted that the relationship between R_e and R_d is influenced by many factors, such as the distribution of the number of intrusions on the distance $d(\vec{v_s}, \vec{v_l})$ and the distribution of the number of behaviors on the distance $d(\vec{v_s}, \vec{v_l})$. Three typical kinds of relationships between R_e and R_d are specified as follows (here, a and b are real numbers):

- $R_e = 1 - ae^{(b-R_d)}$. In this case, intrusion confinement is very necessary since a small improvement in R_d can cause a significant degradation of R_e.
- $R_e = 1 - (a - bR_d)$. In this case, if b is large, then intrusion confinement is necessary; if b is small, then a satisfactory trade-off between R_e and R_d is achievable.
- $R_e = 1 - a\log(b - R_d)$. In this case, a satisfactory trade-off is usually achievable since a small degradation of R_e can cause a significant improvement of R_d.

Of course, there are many systems in which the relationship between R_e and R_d is none of these. However, the SSO can make a sound decision by a similar analysis.

4 How Can Intrusion Confinement Be Enforced?

4.1 Architecture Support

The architecture of an intrusion confinement system from the perspective of information warfare [AJMB97] is shown in Figure 3.

The *Policy Enforcement Manager* enforces the access controls in accordance with the system security policy on every access request. We assume no data access can bypass it. We further assume that users' accesses will be audited in the *audit trail*. For database systems and for transactional file systems, users' accesses on such data objects as tables and files are usually recorded in specific logs maintained for recovery purposes, instead of the audit trail. For simplicity; these logs are not explicitly shown in Figure 3; we assume that they are associated with each data version.

The *Intrusion Detection and Confinement Manager* applies either statistical profile-based detection techniques or signature-based detection techniques, or both to identify

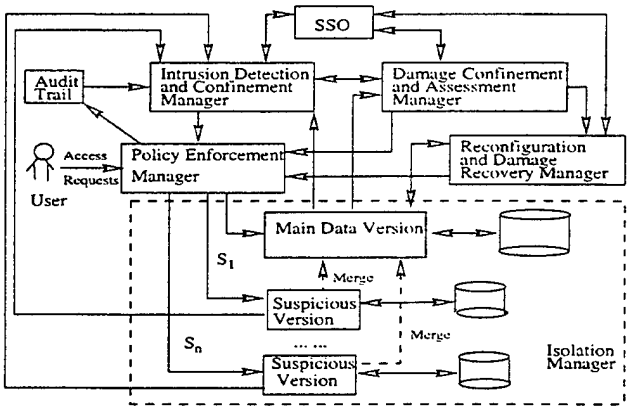


Figure 3. Architecture of the Intrusion Confinement System

suspicious behavior as well as intrusions. The detection is typically processed based on the information provided by the audit trail and the logs associated with each data version.

When a suspicious behavior is detected, the corresponding user is marked *suspicious*. At this point, first we need to deal with the effects that the user has already made on the *Main Data Version* because these effects may have already caused some damage. In signature-based detection systems, we can accept these effects because a partial matching is not an intrusion. In statistical profile-based detection systems, if the SSO does not think these effects can cause any serious damage, we can accept these effects; if the SSO thinks these effects can cause intolerable damage, we can isolate and move these effects from the main data version to a separate *Suspicious Data Version*, which is created to isolate the user. The process of isolation may need to roll back some trustworthy actions that are dependent on the suspicious actions. At this point, we can apply another strategy that moves the effects of these suspicious actions as well as the affected trustworthy actions to the suspicious data version.

Second, the Intrusion Detection and Confinement Manager notifies the Policy Enforcement Manager to direct the subsequent suspicious actions of the user to the separate data version. Since we focus on the isolation itself, we can simply assume that when a suspicious behavior starts to be isolated, no damage has been caused by the behavior. Note that there can be several different suspicious users, e.g., S_1, ..., S_n, being isolated at the same time. Therefore, multiple suspicious data versions can exist at the same time, and these data versions are usually different from each other.

When a suspicious user turns out to be *malicious*, that is, his/her behavior has led to an intrusion, the corresponding suspicious data version can be discarded to protect the main data version from harm. On the other hand, when the user turns out to be innocent, the corresponding suspicious data version is merged into the main data version. A suspicious behavior can be malicious in several ways: (1) In signature-based detection, a complete matching can change a suspicious behavior to malicious. (2) Some statistics of gradual anomaly, such as frequency and total number, can make the SSO believe that a suspicious behavior is malicious. (3) The SSO can find that a suspicious behavior is malicious based on some non-technical evidences.

A suspicious behavior can be innocent in several ways: (1) In signature-based detection, when no signatures can be matched, the behavior proves innocent. (2) The SSO can prove it to be innocent by some non-technical evidence. For example, the SSO can investigate the user directly. (3) Some statistics of gradual anomaly can also make the SSO believe that a behavior is innocent.

Since intrusion confinement cannot isolate every intrusion in most cases (The rate of isolation, R_i, is usually less than 1), intrusions do happen on the main data version. When such an intrusion is detected (this type of intrusion is usually detected by some other approaches beyond the standard mechanisms), the corresponding user is marked as *malicious*. The Intrusion Detection and Confinement Manager then notifies the *Damage Confinement and Assessment Manager* to confine and assess the damage caused by the intrusion. The confinement is done by notifying the Policy Enforcement Manager to restrict the access by the user by either rejecting his/her further access or isolating the damage to prevent further spread. For example, damaged data may be forbidden to be read for a while. Concrete damage confinement mechanisms are out of the scope of the paper.

After the damage is assessed, the *Reconfiguration Manager* reconfigures the system to allow access to continue in a degraded mode while repair is being done by the *Damage Recovery Manager*. In many situations damage assessment and recovery are coupled with each other closely. For example, recovery from damage can occur during the process of identifying and assessing damage. Also, the system can be continuously reconfigured to reject accesses to newly identified, damaged data objects and to allow access to newly recovered data objects. Interested readers can refer to [AJL, LAJ] for more details on damage confinement, damage assessment, system reconfiguration, and damage recovery mechanisms in the database context.

Intrusion confinement mitigates significantly the overhead of damage confinement, damage assessment, system reconfiguration, and damage recovery. This is because sub-

stantial intrusions can be isolated before they happen; thus, they will not cause damage to the main data version. However, we need to pay the extra cost of enforcing intrusion confinement.

Damage recovery may influence the process of intrusion confinement. For example, if a set of actions is found to be malicious and require system recovery, then some actions that are affected by these malicious actions may have already been marked as suspicious and isolated. At this point, removing the effects of these malicious actions from the main data version can lead to removing the effects of these affected suspicious actions from the corresponding suspicious data version. For simplicity and to focus on isolation itself, in the rest of the paper we assume that when a suspicious behavior is isolated there is no damage in the main data version.

4.2 Types of Isolation and Mergers

In the normal course of events when all users are believed to be trustworthy, the data has only one version, namely, the main data version; any updates by a user are seen by all other users of the database and vice versa. When a suspicious user S_i is detected, there are several different types of data flow between the trustworthy actions working on the main data version and the suspicious actions of S_i:

Complete Isolation : Updates by S_i's actions are not disclosed to any trustworthy action, and any updates by a trustworthy action are not disclosed to S_i.

One-way Isolation : Updates by S_i's actions are not disclosed to any trustworthy action, but all updates by a trustworthy action are disclosed to S_i. When an update is disclosed to an action, the action can read the updated value. By one-way isolation, S_i can read the latest value of the data objects kept in the main data version.

Partial Isolation : Some updates by S_i's actions are disclosed to trustworthy actions, and vice versa. This can happen when S_i's updates on some data objects are not considered risky, while anything else S_i attempts to update is confined to the suspicious version. By partial isolation, if some of the updates by trustworthy users are considered sensitive, they could be selectively withheld from being propagated to the suspicious version where they would be divulged to the suspicious user. One drawback of disclosing S_i's updates to trustworthy users is that after S_i is revealed as malicious, some trustworthy actions affected by S_i's updates may have to be backed out.

Data flows among suspicious users can also be grouped into the above three types. However, we have not found many situations where it is useful to disclose updates among suspicious versions, except when the SSO finds that several suspicious users cooperate with each other to do something. At this point, it is helpful that these suspicious users are isolated within one suspicious version because if they are malicious users who collude to do intrusions and to protect themselves from being detected, then isolating each user within a separate version can alert the user to the fact that something is wrong. If they are innocent users, then isolating each user in a separate version can prevent him/her from inter-communicating. However, since collusions are usually difficult to detect, and it is usually a complicated and computing intensive process to ensure the correctness of such data flows. We will not address the problem in this paper and we would like to address it in our future research.

When a user is discovered to be malicious or innocent, a decision must be made on how to accept and merge his/her updates into the main data version. At this point, two types of mergers are possible:

Complete Merger : When a user is discovered to be malicious, all of his/her updates are discarded. When a user is discovered to be innocent, all of his/her updates are considered for merging.

Partial Merger : Remerging of the data versions could be partial. Even if the user were found to be a malefactor, it might be desirable to accept certain of his/her updates into the main database version (for example, after being examined, or those on particular data items) rather than discarding them all outright.

4.3 Identification and Resolution of Conflicts

Because suspicious action histories keep growing, conflicts may arise between trustworthy and suspicious actions. As a result, the main data version and these suspicious data versions may become inconsistent. For example, a trustworthy action and a suspicious action may update the same data object x; thus, neither the main data version of x nor the suspicious data version of x is correct when we decide to merge the suspicious version into the main version.

The techniques to identify and resolve these conflicts usually vary from one type of system to another. For example, the techniques we once proposed for database systems in [JLM98] are quite different from the techniques we will propose for file systems in Section 5. However, we can generally classify these techniques into two categories:

Static Resolution allows both the main action history and the suspicious action histories to grow without any restrictions. Identification and resolution of conflicts are delayed until some suspicious history is designated to be merged into the main history.

Dynamic Resolution does not allow either the main history or the suspicious histories to grow unless the mutual consistency is guaranteed.

4.4 Requirements for an Isolation System

An isolation system should satisfy several general requirements to ensure the security and correctness of intrusion confinement.

Disclosure Property : The isolation strategy should never cause risky data flows from suspicious data versions to the main data version.

Consistency Property : Before and after each merger, the main data version and the suspicious data versions should be kept consistent. The consistency of the main version can be relaxed when some suspicious updates are disclosed to it. The consistency of a suspicious version can be relaxed when some trustworthy updates are disclosed to it.

Synchronization Property : A suspicious data version cannot be merged into the main data version if it is still dependent on some other suspicious data versions.

5 Intrusion Confinement in File Systems

In this section, we will present a concrete isolation protocol in the file system context to evaluate the feasibility of our general intrusion confinement solution.

5.1 Isolation Protocol

Consider a file system with a set of files, denoted f_1, f_2,..., f_n, that are accessed by users. These files can be of many types, such as normal files, directories, and devices. A user can access a file in many ways, such as reading, writing, modifying, renaming, deleting, copying, and moving. In this section, we choose one-way isolation as the isolation strategy and complete merger as the merging strategy, and forbid data flows among suspicious versions.

The isolation protocol which is specified as follows is adapted from [PPR+83], where a protocol is proposed to detect and resolve mutual inconsistency in distributed file systems. In this protocol, the isolation is processed in terms of each file. When a file f_i is modified by a suspicious user S_i, the modification and the possible following modifications of S_i on f_i will be isolated until S_i proves to be malicious or innocent. To identify the conflicts between the modifications of S_i on f_i and the modifications of trustworthy users on f_i, we associate a version vector with the main version of f_i and a version vector with every isolated version of f_i. Note that if a file has never been modified by a suspicious user, then it has no version vectors.

- Each file f_i is associated with a system-wide, unique identifier, called *origin point* (denoted $OP(f_i)$), which is generated when f_i is created. It is an immutable attribute of f_i, although f_i's name is not immutable (indeed, f_i may have different names in different versions). Thus, no number of modifications or renamings of f_i will change $OP(f_i)$.

- At the very beginning when there are no suspicious users, the version vector of the main version of a file f_i can be viewed as $< G : 0 >$, although we do not maintain such a vector until f_i is updated by a suspicious user. Here, G denotes the main version.

- When a file f_i is firstly modified by a suspicious user (denoted S_1), an isolated version of f_i (with the same $OP(f_i)$), called the S_1-version of f_i, is created for S_i to perform the modification. As a result, two different versions of f_i exist after the modification: one is the main version, with no effects of the modification on it; the other is the S_1-version on which the modification is performed. We generate the corresponding version vectors as follows: (1) for the main version, $< G : 0, S_1 : 0 >$ is created as the version vector. $S_1 : 0$ signifies that the version has not been modified by S_1. $G : 0$ signifies that the version has not been modified by trustworthy users since f_i is firstly modified by a suspicious user (here the user is S_1). (2) for the S_1-version, $< G : 0, S_1 : m >$ is created as the version vector. The G dimension ($G : 0$) is copied from the main version vector. $S_1 : m$ means that f_i has been *m*odified by S_1. Note that m is not a number. The S_1 dimension remains to be m no matter how many times the S_1-version is modified.

- The version vector of the main version of a file f_i, if it exists, is changed by modifications performed by trustworthy users as follows: (1) after each such modification, its value in the G dimension is increased by 1, i.e., from $G : n$ to $G : n + 1$. (2) the value in any other dimension is unchanged.

- When a suspicious user S_i asks to modify a file f_i (we assume f_i has already been modified by some suspicious users), if an S_i-version of f_i exists, then the version is given. Otherwise, if there is neither active nor deleted S_i-versions, then we first insert into the main version vector of f_i the S_i dimension with the value $S_i : 0$, then create the S_i-version of f_i on which S_i performs the modification. The S_i-version vector is created by first copying the main version vector and then changing its value in the S_i dimension to $S_i : m$. As before, additional modifications of S_i on f_i do not change the S_i-version vector.

- When a suspicious user S_i asks to delete a file f_i, if there is an S_i-version of f_i that has not been deleted, then (1) the S_i-version is deleted; (2) the value of the S_i-version vector in the S_i dimension is changed to $S_i : d$, which indicates that the S_i-version of f_i is removed. However, the origin point $OP(f_i)$ associated with the S_i-version vector of f_i remains. Otherwise, if there is neither active nor deleted S_i-versions, then the S_i dimension with the value $S_i : 0$ is inserted into the main version vector of f_i, and the S_i-version vector of f_i is created by first copying the main version vector of f_i and then changing its value in the S_i dimension to $S_i : d$.

- When a trustworthy user asks to delete a file f_i, if the main version exists, then the G dimension of the main version vector of f_i (if any) is changed to $G : d$, and the main version of f_i will be removed. However, the origin point $OP(f_i)$ associated with the main version vector of f_i remains. We keep the vector because at this time some suspicious versions of f_i could still exist.

- When a suspicious user S_i asks to access a file f_i and the value of the S_i-version vector in the S_i dimension is d, then the system notifies S_i that f_i does not exist. When a trustworthy user asks to access f_i and the value of the main vector in the G dimension is d, then the system notifies the user that f_i does not exist.

- When a suspicious S_i asks to read a file f_i, if there is a S_i-version of f_i, then the version is given to S_i. Otherwise, if the main version of f_i exists, then the main version is given.

- When a suspicious version is merged with the main version, the main version vector of f_i can have more dimensions than the suspicious version vector of f_i. To identify and resolve the conflicts between these two versions, we need to first pad the suspicious version vector such that the two vectors have the same set of dimensions. The padding is done by inserting each missed dimension with the value 0 into the suspicious version vector. The techniques used to identify and resolve the conflicts are presented in next section.

5.2 Identification and Resolution of Conflicts

There are two types of conflicts that we wish to consider: *name conflicts* and *version conflicts*. A name conflict occurs when two files with different origin points have the same name. In contrast, a version conflict occurs when two versions of the same file (the same origin point) have been incompatibly modified. For example, two versions of a file generated by independent modifications of an older version of the file can introduce a version conflict. Two versions of a file are *compatible* iff there is no version conflict between them.

When a suspicious data version is merged into the main data version, we identify the version conflicts on a file f_i as follows:

- If both the suspicious version of f_i and the main version of f_i are deleted, or at least one of them is deleted, then they are compatible.

- If none of them are deleted, then the two versions are compatible if their corresponding version vectors are compatible. Two version vectors are *compatible* if one vector $\vec{v_i}$ *dominates* the other vector $\vec{v_j}$ in the value of every dimension. If so, we say $\vec{v_i}$ *dominates* $\vec{v_j}$. In our model, value domination is defined as follows: (1) A value dominates itself. (2) A number n_1 dominates another number n_2 if n_1 is larger than n_2. (3) The value m (in $S_i : m$) dominates the value 0. (4) The value d is dominated by any other values.

Name conflicts can be easily resolved by renaming files after all of the version conflicts are resolved. Resolution of version conflicts should be accompanied by resolution of version vector conflicts because we cannot merge two versions without merging their version vectors. When the conflicts between a suspicious S_i-version (with version vector $\vec{v_s}$) and a main data version (with version vector $\vec{v_g}$) of a file f_i are identified, we resolve the conflicts and merge the two versions as follows. The protocol can ensure that in the merged vector: (1) $G : d$ indicates the merged version is removed; otherwise, it exists. (2)$S_i : 0$ indicates there is (or was) a suspicious S_i-version. The version may still be active or may have already been discarded. (3)$S_i : d$ indicates there was a suspicious S_i-version that has been deleted by S_i, who is innocent. (4)$S_i : m$ indicates there was a suspicious S_i-version. The version is modified by S_i and is merged.

- If both of the two versions are deleted, then they need not be merged; however, their version vectors need be merged. The merging is done by taking $\vec{v_g}$ with its value in the S_i dimension changed to $S_i : d$ as the merged vector.

- Suppose that the main version is deleted, but the S_i-version is not deleted. If the value of $\vec{v_s}$ in the S_i dimension is m, then the S_i version is the merged version, and the merged vector is $\vec{v_g}$ with its value in the G dimension changed to that of $\vec{v_s}$ and its value in the S_i dimension changed to m. Otherwise, the deleted main version is the merged version, and the merged vector is $\vec{v_g}$.

- Suppose that the S_i-version is deleted, but the main version is not deleted. If the value of $\vec{v_g}$ in the G dimension is larger than that of $\vec{v_s}$, or there is a dimension such that the value of $\vec{v_g}$ in the dimension is m but the value of $\vec{v_s}$ in the dimension is 0 or $\vec{v_s}$ does not include the dimension, then the main version is the merged version, and $\vec{v_g}$ with its value in the S_i dimension changed to $S_i : d$ is the merged vector. Otherwise, the deleted S_i version is the merged version, and $\vec{v_g}$ with its value in the G dimension changed to d and its value in the S_i dimension changed to d is the merged vector.

- If none of the two versions are deleted:

  - If $\vec{v_g}$ dominates $\vec{v_s}$, then the main version is the resolved version. $\vec{v_g}$ is the merged vector.
  - If $\vec{v_s}$ dominates $\vec{v_g}$, then the suspicious version is the resolved version. $\vec{v_s}$ is the merged vector.
  - If $\vec{v_g}$ and $\vec{v_s}$ are incompatible, then the resolution can be done either manually or automatically, based on the semantics of the modifications that have been applied on these versions. After the resolution, $\vec{v_g}$ with its value in the S_i dimension changed to m is the merged vector. Note that here a version conflict can happen only if the value of $\vec{v_s}$ in the S_i dimension is m.

Example 2 *Assume that when the S_2-version is merged into the main version, the main version vector of a file f_i is: $\vec{v_g}(f_i) =< G : 0, S_1 : 0, S_2 : 0 >$. At this point, if the S_2-version vector of f_i is $\vec{v_s}(f_i) =< G : 0, S_1 : 0, S_2 : m >$, then since $\vec{v_s}(f_i)$ dominates $\vec{v_g}(f_i)$, so there are no conflicts, and the S_2-version is the merged version, and $\vec{v_s}(f_i)$ is the merged vector. If $\vec{v_s}(f_i) =< G : 0, S_1 : 0, S_2 : d >$, then the two versions are still compatible. The deleted S_2-version is the merged version, and $< G : d, S_1 : 0, S_2 : d >$ is the merged vector.*

Consider another scenario where $\vec{v_g}(f_i) =< G : 2, S_1 : 0, S_2 : 0 >$ and $\vec{v_s}(f_i) =< G : 0, S_1 : 0, S_2 : m >$. $\vec{v_g}(f_i)$ and $\vec{v_s}(f_i)$ conflict. Note that the version of f_i with the vector $< G : 0, S_1 : 0, S_2 : 0 >$ has been independently modified by S_2 and some trustworthy transactions. At this point, manual or automatic approaches need be applied to resolve the conflicts. The version vector of the resolved version is $< G : 2, S_1 : 0, S_2 : m >$.

To reduce the overhead of maintaining version vectors, we need to reset main version vectors. The reset for the main version vector of f_i can be processed when there are no active accesses to f_i and when f_i has not been modified or deleted in any active suspicious versions. The reset can be easily done by removing the main version vector.

Although there is no general way to resolve conflicts automatically, conflict resolution can be automated in many file systems by exploiting the operation semantics. For example, reconciliation for two important types of files in LOCUS [PPR+83], directories and user mailboxes, is handled automatically. Interested readers can refer to [PPR+83] for more details on this topic.

6 Related Work

A substantial body of work has been done on intrusion detection [Lun93, MHL94, LM98], based on either detecting deviations from expected statistical profiles [JV94] or pattern-matching against known methods of attack [Ilg93, GL91, PK92, IKP95, SG91, SG97, LWJ98]. In [JV94], the idea of setting multiple alert levels is proposed, where each alert level corresponds to a specific degree of anomaly and different actions are taken at each alert level. However, the issues of what actions should be taken at each level and how to enforce these actions are not addressed in [JV94]. Our isolation scheme can be viewed as a realization of the idea in which two alert levels are set. At the first alert level, we isolate users' accesses when a suspicious behavior is discovered. At the second alert level, we reject users' accesses when an intrusion is reported. Multi-level isolation schemes are certainly possible.

In [LV92] and [HLR92], a probabilistic model of intrusion detection is proposed in which intrusion detection is characterized as an estimation problem. In the model, the probability that a behavior is a signature-based is measured by modeling computer activity as a finite stationary stochastic process. In our probabilistic model, P_i (or R_e) can be measured by their model; however, the *rate of detection* (R_d) is not addressed in their model.

In [JLM98], an application-level isolation protocol is

proposed to cope with malicious database users. In this paper, we extend the work of [JLM98] in several aspects: (1) [JLM98] does not answer clearly the questions such as "why are there suspicious actions," "how can these suspicious actions be detected," "why is isolation necessary," and "when should isolation be enforced." In this paper, we give clear answers to these questions by proposing, modeling, and analyzing the problem of *intrusion confinement* based on a probabilistic model presented to evaluate the effectiveness of current intrusion detection systems. (2) [JLM98] is limited to the database context. In this paper, we extend the isolation mechanisms proposed in [JLM98] to a general solution that can be applied in many types of information systems, such as file systems and object systems. (3) We present a novel isolation protocol for file systems that is not addressed by [JLM98].

Although the area of IW defense is new, some relevant work exists. Graubart et al. [GSM96] identify a number of aspects of the management of a DBMS that affect vulnerability with respect to IW. McDermott and Goldschlag [MG96a, MG96b] identify some techniques for defending against data jamming that, while primarily intended for detection, could also help deceive the attacker and confuse the issue of which data values are critical.

Finally, Ammann et al. [AJMB97] take a detailed look at the problem of surviving IW attacks on databases. They identify a number of phases of the IW process and describe activities that occur in each of them. To maintain precise information about the attack, they propose to mark data to reflect the severity of detected damage as well as the degree to which the damaged data has been repaired. They define an access protocol for normal transactions and show that transactions following the protocol will maintain database consistency even if part of the database is damaged.

7 Conclusion

In this paper, we proposed, modeled, and analyzed the problem of intrusion confinement. It is shown that intrusion confinement can effectively resolve the conflicting design goals of an intrusion detection system by achieving both a high rate of detection and a low rate of errors. It is also shown that as a second level of protection in addition to access control intrusion confinement can dramatically enhance the security (especially integrity and availability) of a system in many situations.

We proposed a general solution that is based on a specific isolation mechanism to achieve intrusion confinement. We evaluated the feasibility of the solution by presenting a concrete isolation scheme that can be enforced in the file system context. It is shown that this protocol is more flexible, economical, and efficient than fishbowling, and it can be applied to every file system. Finally, we would mention that the general intrusion confinement solution can be applied to many other types of information systems in addition to file systems, such as database systems (a simple isolation scheme is proposed in [JLM98]), object-oriented systems, distributed information systems, and workflow management systems. Developing concrete isolation protocols for these systems is a topic of our future research.

References

[AJL] P. Ammann, S. Jajodia, and P. Liu. Recovery from malicious transactions. Technical report, George Mason University, Fairfax, VA. http://www.isse.gmu.edu/~pliu/papers/dynamic.ps.

[AJMB97] P. Ammann, S. Jajodia, C.D. McCollum, and B.T. Blaustein. Surviving information warfare attacks on databases. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 164–174, Oakland, CA, May 1997.

[GL91] T.D. Garvey and T.F. Lunt. Model-based intrusion detection. In *Proceedings of the 14th National Computer Security Conference*, Baltimore, MD, October 1991.

[GSM96] R. Graubart, L. Schlipper, and C. McCollum. Defending database management systems against information warfare attacks. Technical report, The MITRE Corporation, 1996.

[HLR92] P. Helman, G. Liepins, and W. Richards. Foundations of intrusion detection. In *Proceedings of the 5th IEEE Computer Security Foundations Workshop*, pages 114–120, June 1992.

[IKP95] K. Ilgun, R.A. Kemmerer, and P.A. Porras. State transition analysis: A rule-based intrusion detection approach. *IEEE Transactions on Software Engineering*, 21(3):181–199, 1995.

[Ilg93] K. Ilgun. Ustat: A real-time intrusion detection system for unix. In *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, May 1993.

[JLM98] S. Jajodia, P. Liu, and C.D. McCollum. Application-level isolation to cope with malicious database users. In *Proceedings of the 14th Annual Computer Security Application Conference*, pages 73–82, Phoenix, AZ, December 1998.

[JV94] H. S. Javitz and A. Valdes. The nides statistical component description and justification. Technical Report A010, SRI International, March 1994.

[LAJ] P. Liu, P. Ammann, and S. Jajodia. Rewriting histories: Recovery from malicious transactions. *Distributed and Parallel Databases*. To appear. http://www.isse.gmu.edu/~pliu/papers/semrepair.ps.

[LM98] Teresa Lunt and Catherine McCollum. Intrusion detection and response research at DARPA. Technical report, The MITRE Corporation, McLean, VA, 1998.

[Lun93] T.F. Lunt. A Survey of Intrusion Detection Techniques. *Computers & Security*, 12(4):405–418, June 1993.

[LV92] G.E. Liepins and H.S. Vaccaro. Intrusion detection: Its role and validation. *Computers & Security*, (11):347–355, 1992.

[LWJ98] J. Lin, X. S. Wang, and S. Jajodia. Abstraction-based misuse detection: High-level specifications and adaptable strategies. In *Proceedings of the 11th IEEE Computer Security Foundations Workshop*, Rockport, Massachusetts, June 1998.

[MG96a] J. McDermott and D. Goldschlag. Storage jamming. In D.L. Spooner, S.A. Demurjian, and J.E. Dobson, editors, *Database Security IX: Status and Prospects*, pages 365–381. Chapman & Hall, London, 1996.

[MG96b] J. McDermott and D. Goldschlag. Towards a model of storage jamming. In *Proceedings of the IEEE Computer Security Foundations Workshop*, pages 176–185, Kenmare, Ireland, June 1996.

[MHL94] B. Mukherjee, L. T. Heberlein, and K.N. Levitt. Network intrusion detection. *IEEE Network*, pages 26–41, June 1994.

[PG99] Brajendra Panda and Joe Giordano. Reconstructing the database after electronic attacks. In Sushil Jajodia, editor, *Database Security XII: Status and Prospects*. Kluwer, Boston, 1999.

[PK92] P.A. Porras and R.A. Kemmerer. Penetration state transition analysis: A rule-based intrusion detection approach. In *Proceedings of the 8th Annual Computer Security Applications Conference*, San Antonio, Texas, December 1992.

[PPR+83] D.S. Parker, G.J. Popek, G. Rudisin, A. Stoughton, B.J. Walker, E. Walton, J.M. Chow, D. Edwards, S. Kiser, and C. Kline. Detection of mutual inconsistency in distributed systems. *IEEE Transactions on Software Engineering*, 9(3):240–247, 1983.

[SG91] S.-P. Shieh and V.D. Gligor. A pattern oriented intrusion detection model and its applications. In *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, May 1991.

[SG97] S.-P. Shieh and V.D. Gligor. On a pattern-oriented model for intrusion detection. *IEEE Transactions on Knowledge and Data Engineering*, 9(4):661–667, 1997.

Analyzing the Performance of Program Behavior Profiling for Intrusion Detection*

Anup K. Ghosh & Aaron Schwartzbard

Reliable Software Technologies Corporation
21351 Ridgetop Circle, Suite 400 Dulles, VA 20166
POC email: aghosh@rstcorp.com
www.rstcorp.com

Abstract

This paper presents an analysis of a simple equality matching algorithm that detects intrusions against systems by profiling the behavior of programs. The premise for this work is that abnormally behaving programs are a primary indicator of computer intrusions. Program behavior profiles are built by capturing system calls made by the target program under normal operational conditions using Sun Microsystem's Basic Security Module (BSM) auditing facility. Potential intrusions are flagged when a significant portion of BSM events in a given interval are determined to be anomalous. This approach exploits the temporal locality of anomalies induced by attacks to reduce the false alarm rate caused by anomalous noise as well as detect intrusions that might otherwise be washed out in the anomalous noise. The analysis uses data collected by the Air Force Research Laboratory and provided by the MIT Lincoln Laboratory under the 1998 DARPA Intrusion Detection Evaluation program. Attack sessions are embedded in normal background traffic so that the analysis can measure the probability of detection simultaneously with the probability of false alarm. The analysis uses Receiver Operator Characteristic (ROC) curves to show the performance of the system in terms of the probability of false alarm and probability of detection for different operating points. These results, can in turn, be used to tune the system to meet a particular network's requirements on detection and false alarm rates.

*This work is sponsored under Defense Advanced Research Projects Agency (DARPA) Contract DAAH01-98-C-R145. THE VIEWS AND CONCLUSIONS CONTAINED IN THIS DOCUMENT ARE THOSE OF THE AUTHORS AND SHOULD NOT BE INTERPRETED AS REPRESENTING THE OFFICIAL POLICIES, EITHER EXPRESSED OR IMPLIED, OF THE DEFENSE ADVANCED RESEARCH PROJECTS AGENCY OR THE U.S. GOVERNMENT.

1 Introduction

Intrusion detection generally can be defined as a method to detect any unauthorized use of a system. Intrusions are commonly considered to be attacks from unauthorized outside entities to gain access to a target system. Once inside, an intruder can illegally use resources, view confidential material, or even try to subvert or harm the system. The term is less commonly used, yet equally applicable, to describe attacks from within the target system by authorized insiders who attempt to abuse their privileges or acquire higher privileges in violation of security policies.

Intrusion detection techniques are generally classified into one of two approaches: misuse detection and anomaly detection. Misuse detection methods attempt to model attacks on a system as specific patterns, then systematically scan the system for occurrences of these patterns [Garvey and Lunt, 1991, Ilgun, 1992, Lunt, 1993, Kumar and Spafford, 1996, Porras and Kemmerer, 1992]. This process involves a specific encoding of previous behaviors and actions that were deemed intrusive or malicious. Anomaly detection assumes that intrusions are highly correlated to abnormal behavior exhibited by either a user or network traffic. The basic idea of anomaly detection is to baseline the normal behavior of the object being monitored and then flag behaviors that are significantly different from this baseline as abnormalities, or possible intrusions [D'haeseleer et al., 1996, Lunt, 1993, Lunt and Jagannathan, 1988, Lunt, 1990, Lunt et al., 1992, Monrose and Rubin, 1997].

The most significant disadvantage of misuse detection approaches is that they will only detect the attacks that they are trained to detect. Novel attacks or even variants of common attacks often go undetected. In a time when new security vulnerabilities in software are discovered and exploited every day, the reac-

tive approach embodied by misuse detection methods is not feasible for defending systems.

The main advantage of anomaly detection approaches is the ability to detect novel attacks, variants of known attacks, and deviations from normal usage of programs regardless of whether the source is a privileged internal user or an unauthorized external user. One drawback of anomaly detection approaches is that well-known attacks may not be detected, particularly if they fit the established profile of the user. Once detected, it is often difficult to characterize the nature of the attack for forensic purposes. Another drawback of many anomaly detection approaches is that a malicious user who knows he or she is being profiled can change his or her profile slowly over time to essentially train the anomaly detection method to learn his or her malicious behavior as normal. Finally, a narrowly trained detection algorithm may result in a high false positive rate, or conversely, a broadly trained anomaly detection approach may result in a high false negative rate.

Desiring the ability to detect novel, unknown attacks against systems, an anomaly detection approach is employed in this research. Unlike traditional anomaly detection approaches, this research applies anomaly detection at the system process level. Rather than profiling the normal behavior of users, the approach here is to profile the normal behavior of executing processes. Monitoring at the process level abstracts away users' idiosyncrasies such that abnormal process behavior can be detected irrespective of individual users' behavior. Having baselined the normal behavior of a process, deviations from the normal behavior can be used to detect attempted exploitations of the program. A corollary of this approach is that attempted misusage of programs that does not alter normal program behavior is not flagged.

2 Prior art

While intrusion detection, and anomaly detection approaches in particular, have been the focus of much recent research, the work of two groups out of the University of New Mexico and Columbia University is most closely related to ours and has been leveraged for the purposes of our research.

The work of Forrest et al. from the University of New Mexico has firmly established the analogy between the human immune system and intrusion detection [Forrest et al., 1997, D'haeseleer et al., 1996, Forrest et al., 1996]. The UNM work was among the first to focus intrusion detection at the system process level (in addition, see [Voas et al., 1992]). The UNM group used short sequences of system calls from the target program to the operating system to form a signature for normal behavior. A database of system calls is built from executing a program under normal usage conditions and capturing all system calls made by the program. Once constructed, the database essentially serves as the repository for self behavior against which all subsequent online behavior will be judged. If a string formed during the online operation of the program does not match a string in the normal database, a mismatch is recorded. If the number of mismatches detected is a significant percentage of all strings captured during the online session, then an intrusion is registered. This approach has the drawback that anomalous events are averaged out over the entire execution trace, which could result in attacks being washed out in anomalous noise (see Section 4).

The work of the Columbia group applied a rule-based expert system for detecting intrusions to the data collected by the UNM group [Lee et al., 1997]. The approach applied a rule learning program, RIPPER [Cohen, 1995], to the data to extract rules for predicting whether a sequence of system calls is normal or abnormal. Because the rules made by RIPPER can be erroneous, a post-processing algorithm is used to scan the predictions made by RIPPER to determine if an intrusion has occurred or not. The post-processing algorithm uses the notion of temporal locality to filter spurious prediction errors from intrusions which should leave temporally co-located abnormal predictions. This notion of temporal locality of abnormal traces is used in our work as described in Section 4. That is, rather than averaging the number of abnormal system calls out of the total number of system calls made (*à la* UNM work), a number of abnormal local regions (consisting of windows of system calls) is used to determine if an intrusion has occurred.

One drawback of rule-based approaches is that it is difficult to encode a specific sequence of actions efficiently because clauses in the `if-then` rules are inherently unordered. Thus, most rules will fire due to an arbitrary permutation of its clauses [Kumar and Spafford, 1996]. RIPPER only outputs rules for the "minority" class. Thus, if the training data has fewer abnormal sequences than normal ones, then the RIPPER rules become a technique for detecting known intrusions (viz., misuse intrusion detection). Conversely, if normal behavior forms the minority class, then RIPPER rules can be used for anomaly detection. The results in [Lee et al., 1997] show the ability to use RIPPER for either anomaly detection or misuse detection.

3 Building program behavior profiles

The data collected for building program behavior profiles is the set of BSM events made by the programs under analysis and recorded by Sun Microsystem's Basic Security Module (BSM) system auditing facility for Solaris. The data was collected over a period of weeks in a controlled environment on an internal Air Force Research Laboratory (AFRL) network and provided to us by MIT's Lincoln Laboratory under the DARPA 1998 Intrusion Detection Evaluation program[1]. Attack sessions are embedded within normal background traffic sessions, which allows us to measure both correct detection and false alarm rates simultaneously. In addition to BSM data, TCP/IP connections were provided through the `tcpdump` network packet sniffer facility. This data was not used in this study as we are attempting to use only program behavior profiles to detect intrusions against these programs. The research described in [Forrest et al., 1997] indicates that short sequences of system calls provide a compact and adequate "signature of self" that can be used to distinguish between normal ("self") behavior and abnormal (dangerous "non-self") behavior. The conclusions in [Forrest et al., 1997] show that the data is still too inconclusive to determine if equality matching of system calls is a viable approach to intrusion detection. The detailed analyses performed in this paper in a controlled, scientific study coordinated with MIT's Lincoln Laboratory demonstrate the viability of this approach when extended to exploit temporal locality.

The session data provided by Lincoln Laboratory was labeled for training purposes such that intrusive sessions can be distinguished from normal sessions. Using these session labels, the intrusion sessions are separated from normal sessions. Because our approach is based on anomaly detection, we use only the normal sessions for training. A database is constructed for each program. The database is simply a table that consists of every unique N-gram of BSM events that occurred during any execution, as well as a frequency count for each N-gram. An N-gram is a sequence of N events. For instance, given the following snippet from a pre-processed file of BSM events (where single letters represent events),

```
[pid 1] A;[pid 1] B;[pid 2] C;[pid 1] D;
[pid 1] B;[pid 2] A;[pid 1] D;[pid 1] B;
[pid 2] E;[pid 2] B;[pid 2] D;,
```

and given an N-gram size of two, a table of all observed N-grams could be constructed. First, the two processes represented in the file must be deinterleaved, producing the following execution traces:

```
pid 1: A B D B D B
pid 2: C A E B D.
```

A sliding window with a size of two (as specified above) would be passed over the execution trace of the first process, to collect all N-grams with a size of two. As a result, the following table would be created. The

BSM N-grams	Frequency
A B	1
B D	2
D B	2

Table 1: **Table of normal behavior profile for the first process for an N-gram size of two. Total number of unique N-grams is three, and the total number of N-grams is five.**

same process is applied to the execution trace of the second process, and the results are added to the same table to produce table 2.

BSM N-grams	Frequency
A B	1
B D	3
D B	2
C A	1
A E	1
E B	1

Table 2: **Table of normal behavior profile for a program for an N-gram size of two. Total number of unique N-grams is six, and the total number of N-grams is nine.**

The table shows the unique strings that are created from a program's BSM profile with their associated frequency. Each table characterizes the normal behavior of the program under non-intrusive usage. Tables such as this one are built programmatically for all programs under analysis. In this study, tables for 155 different UNIX programs were constructed. The tables are used by the intrusion detection algorithm to detect anomalous behavior.

A single day of data captured by the BSM auditing facility produces on the order of hundreds of megabytes of data. This data is then processed to create the tables. Because only unique strings are stored in the tables, the amount of data stored in a table is much smaller than the size of the data that is processed — typically three orders of magnitude in

[1] See `www.ll.mit.edu/IST/ideval/index.html` for a summary of the program.

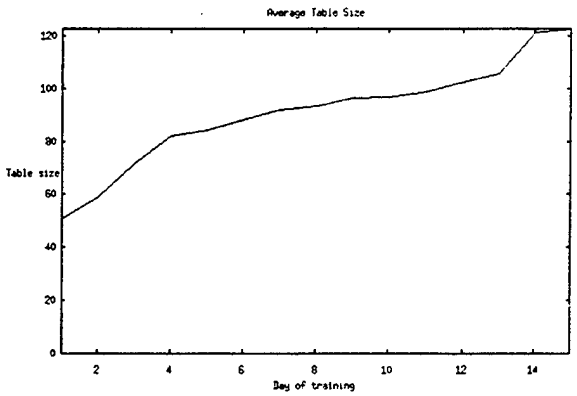


Figure 1: Average number of N-grams in the database for a program. This plot shows how the database grows during the first three weeks of training. If training stops when the tables are very small, legitimate system usage will have high levels of anomalous noise. If training proceeds for too long, many anomalous strings will be entered (and thereafter be considered non-anomalous), resulting in a reduced ability to detect anomalous behavior.

reduction in size. As data continues to be processed day after day, the number of new unique entries in the table drops off significantly. However, because computational resources and storage are limited, it is important to determine at what point the amount of training performed is good enough. In order to determine a suitable stopping point in training, a plot of the amount of data processed in training (in days of training data) versus the average table size (*i.e.*, the number of unique N-grams in a table) is shown in Figure 1. It is clear that after the first week of training data, the rate of increase in average table size slows significantly. Training could stop at this point. However, N-grams do continue to be added to the database after two weeks of training data. In the case of the data shown here, training on day 14 shows another jump in the number of N-grams added to the database.

4 Exploiting temporal locality of attacks

Our approach to intrusion detection is based on the idea that intrusions can be detected by observing the behavior of individual programs. Anomalous behavior of a program could indicate that the program is being subverted for intrusive purposes. However, classifying the behavior of a program as anomalous or non-anomalous for the purpose of detecting intrusions can be rife with errors because program behavior is variable. Using a low threshold for anomalous behavior can result in a high false alarm rate where a non-intrusion is classified as an intrusion (a false positive). High false alarm rates will diminish the utility of the system very quickly. A high threshold for anomalous behavior might result in reduced false alarms, but in a high missed detection rate (that is, a high rate of false negatives). Complicating the matter further, programs will behave differently in different environments. Therefore, there are some behaviors a program might display that may not appear to be normal, but are not subversive, either. This range of behavior results in what might be thought of as *anomalous noise.* In this section, an approach for distinguishing intrusive behavior from anomalous noise is developed.

In order to avoid a large number of false positives, it is necessary for an anomaly-based intrusion detection system to be able to distinguish anomalous noise from intrusive behavior. The approach employed here uses the temporal locality of anomalous sequences characteristic of programs under attack to distinguish intrusive behavior from simple anomalous noise. Figure 2 illustrates the problem of averaging anomalous behavior over the duration of an execution trace.

In Figure 2A, the anomalous noise of the program under analysis is on average less than the threshold, T. From this behavior, a rule might be derived that classifies a session as intrusive if more than $T\%$ of its events are judged to be anomalous. However, this approach has its drawbacks as illustrated in Figures 2B and 2C. A program being used legitimately may occasionally generate $(T + \epsilon)\%$ anomalous noise, as illustrated in Figure 2B, resulting in frequent false positives. Further, an intruder might try to minimize the anomalous noise during an extended use of a program in order to balance several highly anomalous, subversive actions. Thus, the intruder could intentionally slip under the $T\%$ threshold for the program's average anomalous behavior to avoid detection, as illustrated in Figure 2C.

In order to address this problem, the temporal locality of anomalous sequences characteristic of attacks against programs can be exploited. When a program is being misused at the user level of abstraction, the program will make a sequence of anomalous system calls. That is, program misusage tends to result in sequences of anomalous system calls as in Figure 2C. Averaging the number of anomalous sequences of the entire execution trace can result in the intrusion being washed out in the anomalous noise. Thus, it is essential to look for temporally co-located anomalous sequences by dividing the execution trace into fixed-

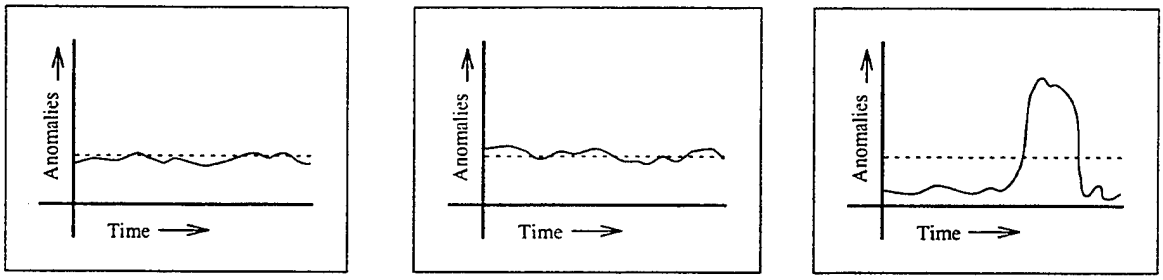


Figure 2: Possible usages of a program in terms of anomalous behavior. If some threshold T (displayed as the dotted line) is applied to the entire execution of a program, false positives and false negatives may occur frequently. A) During a normal execution, the anomalous noise is, on average, less than T. B) Due to normal statistical variance, the anomalous noise will, on average, slightly exceed T, resulting in a false positive. C) A knowledgeable intruder could minimize the anomalous noise of a program (by performing very mundane actions) so that, on average, the subversive behavior does not exceed T, resulting in a false negative.

size intervals of temporal locality that are reflective of macro-level user events. Because each interval has a fixed size much smaller than the overall execution trace, it is possible to avoid *washing out* a few highly anomalous actions over long program executions. Using this approach, it would be desirable for the program analyzed in Figure 2C to choose a fixed interval size on the order of the size of the disturbance shown in the figure.

5 Using equality matching for intrusion detection

The algorithm we use for detecting intrusions is simple but effective. With an approach similar to the that of [Forrest et al., 1997], we employ equality matching of sequences of system calls captured during online usage against those stored in the database built from the normal program behavior profile. If the sequence of BSM events captured during online usage is not found in the database, then an anomaly counter is incremented. This technique is predicated on the ability to capture the normal behavior of a program in a database. If the normal behavior of a program is not adequately captured, then the false alarm rate is likely to be high. On the other hand, if the normal behavior profile built for a program includes intrusive behavior, then future instances of the intrusive behavior are likely to go undetected. Finally, it is important that the range of possible behavior patterns is much larger than the range of normal behavior patterns, because detection of intrusive events is only made possible when the intrusive behavior patterns are different from the normal behavior patterns. The more compact the representation of normal behavior and the larger the range of possible behavior, the better detection capacity this equality matching technique will have. The ratio of normal behavior patterns to possible behavior patterns provides an indication of the detection capacity of the technique. The smaller this ratio, the better the detection capacity.

Program	# of Unique BSM Events
`cat`	7
`in.telnetd`	14
`login`	17
`mail`	13
`pt_chmod`	10
`quota`	7
`sendmail`	18
`sh`	10
`tcsh`	13

Table 3: **Number of unique BSM events for programs run in a sample `telnet` session.**

Capturing system calls via the BSM provides a broad range of possible behavior. There are approximately 200 different BSM events that can be recorded. The N-grams described in Section 3 represent a sequence of N consecutive BSM events. Therefore a single N-gram can have 200^N possible combinations. In practice, however, a program will normally make only about 10 to 20 different system calls. Table 3 shows the number of different BSM events for programs in a sample `telnet` session under non-intrusive conditions.

Using 20 BSM events as an example, this gives rise to 20^N possible N-grams during normal operation. The ratio between possible BSM events during normal behavior to all possible system calls is $1/10^N$. Thus,

as N increases, the ratio of normal BSM events to possible BSM events decreases as a power of 10. In practice, the number of N-grams produced by a program is much smaller than 20^N, mainly because either certain orderings never occur *e.g.*, {exit execve}, or because certain orderings do not correspond to what a particular program would ever do. Our results find that for $N = 6$, each program makes approximately 100 to 200 different N-grams, rather than 20^6. Thus, the ratio of normal sequences of BSM events to possible sequences of BSM events is much smaller in practice than our pessimistic scenario.

Interval	Unit Contained	Threshold
session	file	ST
file	window	FT
window	N-gram	WT
N-gram	BSM event	—

Table 4: **Using intervals of fixed sizes and thresholds to exploit temporal locality of attacks.**

The approach employed here exploits temporal locality of anomalous sequences for detecting intrusions. That is, rather than averaging the number of anomalous sequences out of the total number of sequences, we employ thresholding functions in fixed-size intervals, where the fixed-size interval is much smaller than the total execution trace. A session is considered to be intrusive if it contains an anomalous execution of a program. A program is considered anomalous if a portion of its windows is anomalous. A window is considered anomalous if a portion of its N-grams is anomalous. An N-gram is considered anomalous if it cannot be found in the database corresponding to the program currently being checked.

Table 4 shows the fixed-size intervals, the units they contain, and their thresholds that are used to detect anomalous behavior. When the threshold for a fixed interval — such as a window of N-grams — is exceeded, the whole interval is marked as anomalous. Then a thresholding function is applied at the next level up in abstraction using the marked anomalous interval to determine if an intrusion has occurred. For instance, if a window is marked as anomalous because its threshold, WT, for anomalous N-grams is exceeded, then this window is used as part of the count for file intervals to determine if the file threshold FT is exceeded. Ultimately, the session threshold, ST, must be exceeded for the session to be labeled as an intrusion. The thresholds are all configurable and the tuning of the thresholds will impact the performance of the system as shown in the following sections.

5.1 Tuning parameters of the system

The performance of the system is highly dependent on several independent parameters. Adjusting any one of several parameters can result a large change in the performance of the system, and there is some trade-off involved with each parameter. The parameters are:

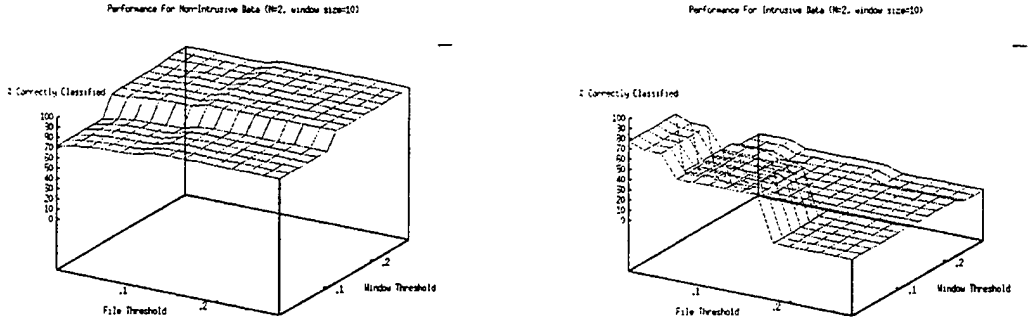
- **Extent of Training** The amount of data used for training is constrained both by processing ability (as training the system is computationally very expensive), and the amount of training data available. The ideal amount of training results in a set of N-grams being collected which characterizes the behavior of a program under normal operation. Too little training results in very high levels of anomalous noise; too much training results in a decreased ability to recognize truly anomalous events as anomalous.

- **N-gram Size** When N is very small, less temporal information is available. In the case where N is one, all temporal information is lost. A very large value for N results in N-grams that carry too much information. Each N-gram should be a general sequence of events that does not necessarily reflect the context in which it occurs. As N-grams carry more contextual information, anomalous noise rises.

- **Window Size** The window size is the number of N-grams in a window. A window is meant to contain a relatively high-level user event. If the window is too small, it might not be able to contain an entire event, and short bursts of anomalous N-grams, which should be averaged into the anomalous noise, extend through the entire window, resulting in the window being falsely classified as anomalous. If the window size is very large, it might encompass many N-grams not related to a subversive action occurring during the window, and thus the subversive action might blend into the anomalous noise.

- **Window Threshold** This threshold is the value that the ratio of anomalous N-grams in a window to total N-grams in a window must exceed before the window is flagged as anomalous.

- **File Threshold** This threshold is the value that the ratio of anomalous windows in the execution of a program to total windows in a program must exceed before the execution of the program is considered to be anomalous.



A B

Figure 3: Performance of the equality matching intrusion detection system with a window size of 10 and an N-gram size of two. These graphs display the percentage of data that is correctly classified when the data is A) non-intrusive, and B) intrusive. As the file and window threshold ratios rise, the system becomes more tolerant of anomalous behavior. Thus, lower thresholds result in an increased ability to detect intrusions, but also a greater likelihood of misclassification of non-intrusions.

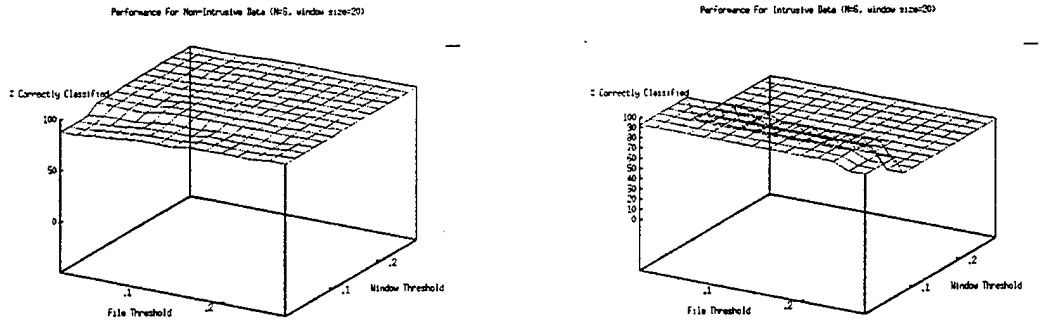
- **Session Threshold** This threshold is the value above which the score for a session must exceed to be flagged as intrusive. This score corresponds to the anomalous measure of the most anomalous file.

The extent of training is typically limited by the amount of data and computational resources on hand. As shown in Figure 1, the amount of new N-grams added to program databases leveled off on average after adding one week of training data. Surprisingly, even with only two weeks of training data, the equality matching system was able to detect intrusions with fairly high accuracy.

Varying the N-gram size and the window size will impact the performance of the system. Various N-gram sizes tested ranged from two to 50. The anomaly detection capabilities drop sharply for values of N larger than 12. Therefore, we focused on smaller values for N. The smallest window size tested consisted of 10 N-grams.

To optimize the file and window thresholds, the performance of the system was measured against these two variables while fixing the N-gram size and the window size. Figure 3A shows the performance of the system with respect to non-intrusive, normal data. As expected, as we increase the window and file thresholds, the performance of the system in classifying non-intrusive data improves. In other words, increasing these thresholds increases the tolerance for anomalous noise in a session. Figure 3B shows that increasing these thresholds for intrusion data decreases the performance of the system. That is, as the tolerance for anomalous noise is increased, the rate of correct classifications will decrease due to missed intrusions (false negatives). Notice in both these plots that the (0.1, 0.1) threshold appears to be a turning point where performance sharply changes. This is the optimum threshold point given the fixed parameters. Plots such as these are used to determine optimal thresholds for tuning the equality matching intrusion system.

To determine optimal N-gram size and optimal window size, plots similar to Figure 3 were constructed except one parameter was fixed, while the other was varied. For instance, to determine optimal N-gram size, the window size was fixed at 20, and numerous plots similar to Figure 3 were constructed for values of N ranging from two to 12 in increments of two. Using this process for both N-gram sizes and window sizes, the optimal performance was found for $N = 6$ and window size of 20. Figure 4 shows the performance of the equality matching system with these optimal parameters under varying file and window thresholds. Figure 4A shows the large plateau in the center of the plane, which indicates that there is a large range of window and file thresholds which produce optimal performance. Figure 4B also shows a well-defined and broad region of good performance for intrusion data. Having a large and well-defined range of performance for these thresholds is important because these thresholds may be varied for different programs and in different environments.



A B

Figure 4: Performance of the equality matching intrusion detection system for optimal window size of 20 and optimal N-gram size of six. These graphs display the percentage of data that is correctly classified when the data is A) non-intrusive, and B) intrusive. The large plateau in the center of these plots indicates good performance over a large range of file and window thresholds.

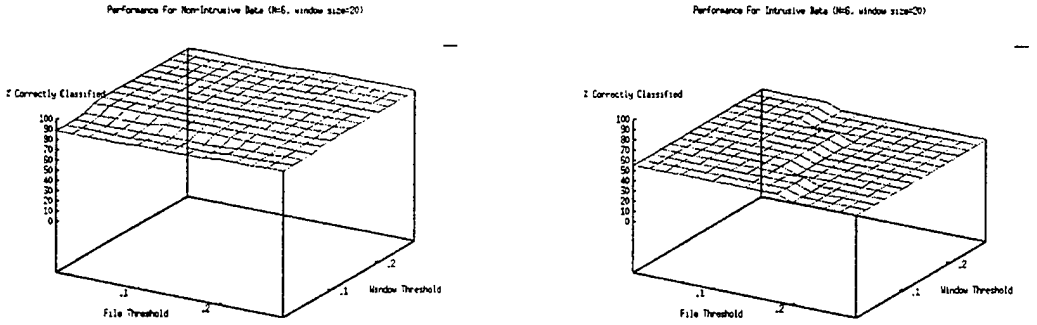
In order to insure that the optimal parameters found were generally applicable, and not simply specific to the data being used, this process was repeated for different training sets and intrusive and non-intrusive sessions. The results from performing this analysis on other data sets showed consistently good performance in correctly classifying non-intrusive data, but varying performance in classifying intrusion data. This result is expected because the ratio of intrusive data to non-intrusive data in the test sets is very small. Thus, one would expect the performance of the system to vary more widely when presented the rare intrusion data. Nonetheless, the results showed that the N-gram size of six and window size of 20 were in fact optimal for the different data sets. Results from testing the system on new data sets are shown in Figure 5.

6 Computing the probability of detection versus the probability of false alarm

In order to judge the effectiveness of this intrusion detection system, it is useful to compute the probability of detection against the probability of false alarm for different operating characteristics, such as the tunable thresholds. Previous evaluations of intrusion detection systems tended to focus exclusively on probability of detection while ignoring the probability of false alarm. False alarms are the Achilles heal of most intrusion detection systems. The higher the false alarm rate, the less effective the system is. If the intrusion detection system cries wolf too frequently, then the user will abandon the system completely. The data provided by MIT's Lincoln Laboratory embedded attack sessions within normal sessions, which permits measurement of detection and the false alarm rates simultaneously.

A measure of the overall effectiveness of a given intrusion detection system can be provided by the receiver operating characteristic (ROC) curve. An ROC curve is a parametric curve that is generated by varying the threshold of the *intrusive measure*, which is a tunable parameter, and computing the probability of detection and the probability of false alarm at each operating point. The curve is a plot of the likelihood that an intrusion is detected, against the likelihood that a non-intrusion is misclassified for a particular parameter, such as a threshold. The ROC curve can be used to determine the performance of the system for any possible operating point. Thus, it can be applied to different file and window thresholds to judge performance. ROC curves were generated for our system for the labeled training data and are illustrated in Figure 6 and Figure 7. The results are for data that the system had not seen before, *i.e.*, data on which it had not trained.

Regardless of the intrusion detection system used, two points remain fixed on the ROC curve: (0,0) and (1,1). The (0,0) point is achieved when the threshold is set to 1, in which case no intrusions will be detected because more than 100% of the intervals used for anomaly detection would have to be anomalous to be considered intrusive. This threshold gives a 0% rate of correct detection of intrusions, and no false alarms.



A B

Figure 5: Performance of the intrusion detection system with a window size of 20 and an N-gram size of six on new data. The variation in performance in intrusive data results from the fact that testing is performed on much more normal data than intrusive data. Thus, normal statistical variation have a larger effect on the intrusive data.

The (1,1) point is achieved when the threshold is set to 0 (inclusive), which means that all sessions will be classified as intrusions giving perfect detection of intrusions and a 100% false alarm rate. The power of the intrusion detection system can be measured by the area under the curve for a given intrusion detection system between these points.

To better understand this performance measure, consider an intrusion detection oracle that scores a session with a value of one if and only if it is an intrusion, and a value of zero otherwise. The resulting ROC curve would actually not be a curve, but rather, a single point at the location (0,1) since it is would detect intrusions with a likelihood of 1/1, and it would misclassify non-intrusions with a likelihood of 0/1. Further, as the threshold varied between zero and one (exclusive), there would be no change in the way sessions are classified, so the parametric value would remain at that one point. This can be called the *oracle point.* However, at the thresholds of 1 and 0 (inclusive), the (0,0) and (1,1) points remain fixed. Connecting these points and computing the area under the curve gives an area of 1, or a power of 100%.

At the other end of the spectrum, consider the curve that defines the worst possible intrusion detection system. The ROC curve for the worst case scenario is the $y = x$ line shown in Figure 6. Assume a system that randomly assigns a value between zero and one for every session. Starting from a threshold of zero, we derive the (1,1) point because all sessions would be classified as intrusions. As the session threshold increases, the likelihood of both correctly classifying an intrusion and incorrectly classifying a non-intrusion decrease at the same rate until the session threshold is 1 (corresponding to the point (0,0)). The power of this system is 50%, corresponding to the area under this curve of 0.5. If an intrusion detection system were to perform even worse than this curve, one would simply invert each classification to do better. Therefore, the $y = x$ plot represents the benchmark by which all intrusion detection system should do better.

The ROC curve allows the end user of an intrusion detection system to assess the trade-off between detection ability and false alarm rate in order to properly tune the system for acceptable tolerances. For example, Figure 6 shows ROC curves produced with a window size of 20 and an N-gram size of six. These parameters were found to be optimal from our previous analyses. In the ROC curves plotted in Figure 6 and Figure 7, the window threshold was held constant for each curve, while the *session threshold* was varied. The session threshold is the value above which the score for a session must exceed to be flagged as possibly intrusive. Recall that the score reported for a session is the measure of the *most* anomalous file in that session.

Three ROC curves are shown in Figure 6. The $y = x$ curve is shown as the benchmark for the worst case scenario. The top curve corresponds to a window threshold fixed at zero, and the other corresponds to a window threshold fixed at 0.2. The ROC curve for the window threshold of 0.2 encloses an area of 0.85. The ROC curve for the window threshold of 0 encloses an area of 0.91. No value for the window threshold re-

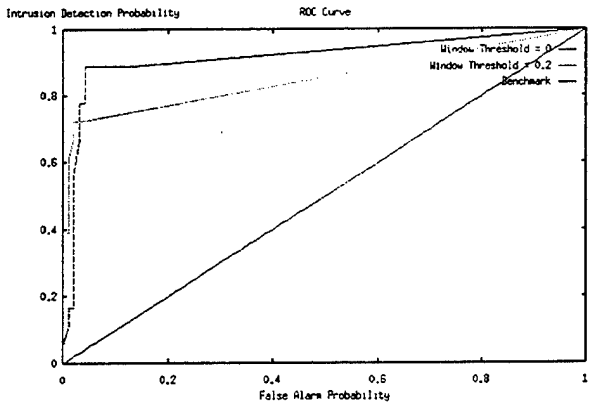


Figure 6: Receiver operating characteristic (ROC) curve for optimal parameters after two weeks of training data. The ROC curve allows an intrusion detection system user to adjust the sensitivity of the intrusion detection system. This graph shows both the worst possible ROC curve (*i.e.*, $y = x$), as well as a ROC curve generated from actual data.

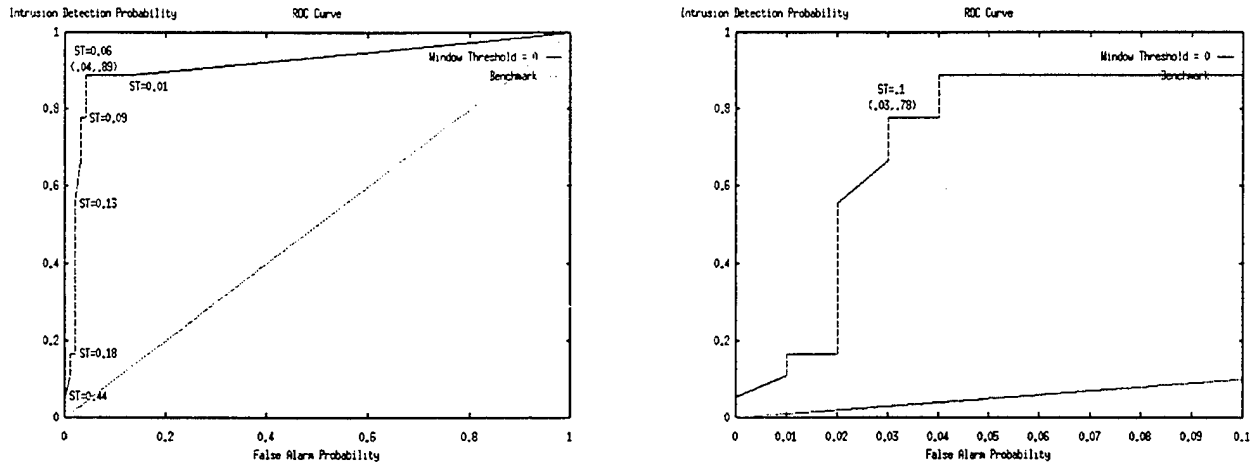
sulted in an enclosed area greater than 0.91. Thus, a window threshold of 0 is optimal. However, there is a range of possible values for the session threshold on this optimal curve. Since the oracle point defines the ideal intrusion detection ability, the point on the curve closest to the oracle point results in the optimal sensitivity. Figure 7A shows the ROC curve for the window threshold of 0, with several points labeled with the session threshold value to which they correspond. It is clear that the point at (0.04, 0.89), which corresponds to a session threshold of 0.06, is closest to the oracle point. In many cases, the ability to detect 89% of all intrusions and produce a false alarm rate only 4% is adequate. In some systems, this performance may not be adequate. For instance, a false alarm rate of 4% might be too high. A difficult problem then arises: how should the sensitivity of the system be adjusted, taking into account that by lowering the sensitivity, one not only decreases the number of false alarms, but also decreases the number of intrusions detected. The scaling of the axes plays an important role in solving this problem.

By appropriately scaling the axes, it is possible to once again reduce the problem of selection of a session threshold to finding the distance to the oracle point. A site that cares more about reducing false alarms than detecting all intrusions should stretch the false-alarm axis (x-axis). If detection of all intrusions is very important, then the intrusion-detection axis (y-axis) should be scaled appropriately. The ROC curve in Figure 7B is the same as that in Figure 7A, however, the axes have been scaled for a site that places greater emphasis on false alarm reduction. In this case, the point on the curve closest to the oracle point is (0.03, 0.78), and occurs when the session threshold is 0.1. Thus, it is possible to achieve a lower false alarm rate (3%) at a cost of reducing the intrusion detection rate (78%) by using a window threshold of 0 and a session threshold of 0.1.

7 Discussion and future work

The analyses presented here are for perhaps the simplest intrusion detection algorithm — equality matching — for detecting intrusions by profiling program behavior. The real constraints of this approach are the computing and storage capacity necessary to process data, and the necessity for good training data in the first place. All experiments described here used two weeks of BSM audit data for training. The benefit of this approach is that BSM is an auditing facility widely available on Solaris operating systems. Therefore, no special programs need to be written to capture the data from which program behavior profiles are built. However, because training and testing data come from the same finite pool of data, and since once data is used for training, it cannot be used for testing, it is difficult to produce significant test results when most of the available data has been used for training. Therefore, our ability to determine the optimal extent of training was limited by these constraints. However, even with two weeks (which is almost certainly sub-optimal), we were able to produce results that indicated that the simple equality matching technique for program behavior profiling provides sufficiently good performance for intrusion detection.

Future directions of this work will extend the intru-



A B

Figure 7: ROC curve with scaled axes. By scaling the axes, the task of determining a session threshold reduces to minimizing the distance to an "oracle point" (in this case, that is (0,1)). Again, both the worst possible curve, and an actual ROC curve are shown.

sion detection algorithm in two similar, but different directions. A logical extension of the equality matching approach is to say that an N-gram is less anomalous if it *resembles* other strings that have been seen before, and more anomalous if it does not resemble any of those strings. We intend to implement two different algorithms that use this notion. First, we will use data interpolation with distance functions to statistically determine how close a given N-gram matches the normal behavior profile. Similar approaches have been successful in areas like density estimation and nonparametric regression. We expect this approach to reduce the number of false positives due to anomalous noise. Second, we are training neural networks to make connections between normal data, and conversely, intrusive data. Using neural networks, we aim to improve the online performance of our system.

Acknowledgment

We are pleased to acknowledge Richard Lippmann and Marc Zissman of MIT's Lincoln Laboratory for providing us the training and testing data, as well as introducing us to the ROC assessment approach.

References

[Cohen, 1995] Cohen, W. (1995). Fast effective rule induction. In *Machine Learning: Proceedings of the Twelfth International Conference.* Morgan Kaufmann.

[D'haeseleer et al., 1996] D'haeseleer, P., Forrest, S., and Helman, P. (1996). An immunological approach to change detection: Algorithms, analysis and implications. In *IEEE Symposium on Security and Privacy.*

[Forrest et al., 1997] Forrest, S., Hofmeyr, S., and Somayaji, A. (1997). Computer immunology. *Communications of the ACM*, 40(10):88–96.

[Forrest et al., 1996] Forrest, S., Hofmeyr, S., Somayaji, A., and Longstaff, T. (1996). A sense of self for unix processes. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pages 120–128. IEEE.

[Garvey and Lunt, 1991] Garvey, T. and Lunt, T. (1991). Model-based intrusion detection. In *Proceedings of the 14th National Computer Security Conference.*

[Ilgun, 1992] Ilgun, K. (1992). Ustat: A real-time intrusion detection system for unix. Master's thesis, Computer Science Dept, UCSB.

[Kumar and Spafford, 1996] Kumar, S. and Spafford, E. (1996). A pattern matching model for misuse intrusion detection. The COAST Project, Purdue University.

[Lee et al., 1997] Lee, W., Stolfo, S., and Chan, P. (1997). Learning patterns from unix process execution traces for intrusion detection. In *Proceedings of AAAI97 Workshop on AI Methods in Fraud and Risk Management.*

[Lunt, 1990] Lunt, T. (1990). Ides: an intelligent system for detecting intruders. In *Proceedings of the Symposium: Computer Security, Threat and Countermeasures.* Rome, Italy.

[Lunt, 1993] Lunt, T. (1993). A survey of intrusion detection techniques. *Computers and Security*, 12:405–418.

[Lunt and Jagannathan, 1988] Lunt, T. and Jagannathan, R. (1988). A prototype real-time intrusion-detection system. In *Proceedings of the 1988 IEEE Symposium on Security and Privacy.*

[Lunt et al., 1992] Lunt, T., Tamaru, A., Gilham, F., Jagannthan, R., Jalali, C., Javitz, H., Valdos, A., Neumann, P., and Garvey, T. (1992). A real-time intrusion-detection expert system (ides). Technical Report, Computer Science Laboratory, SRI Internationnal.

[Monrose and Rubin, 1997] Monrose, F. and Rubin, A. (1997). Authentication via keystroke dynamics. In *4th ACM Conference on Computer and Communications Security.*

[Porras and Kemmerer, 1992] Porras, P. and Kemmerer, R. (1992). Penetration state transition analysis - a rule-based intrusion detection approach. In *Eighth Annual Computer Security Applications Conference*, pages 220–229. IEEE Computer Society Press.

[Voas et al., 1992] Voas, J., Payne, J., and Cohen, F. (1992). A model for detecting the existence of software corruption in real time. *Computers and Security Journal*, 11(8):275–283.

Integrating Data Mining Techniques with Intrusion Detection Methods

Ravi Mukkamala Jason Gagnon
Department of Computer Science
Old Dominion University
Norfolk, VA 23529-0162
mukka@cs.odu.edu

Sushil Jajodia
Department of Information and Software Systems Engineering
George Mason University
Fairfax, VA 22030-4444
jajodia@gmu.edu

Abstract

Intrusion detection systems like NIDES depend on the ability to characterize a user's past behavior based on his/her usage patterns. The characterization is typically made in terms of statistics drawn on system parameters such as CPU, I/O and network loads, and file access patterns. For example, NIDES maintains statistics on approximately 25 such parameters for each user. The cost of data collection, statistics computation, and intrusion detection are directly proportional to the number of parameters maintained per user. If we would like to achieve real-time responses to intrusion detection, then we need to minimize the number of parameters without adversely affecting the detection capabilities.

In this paper, we propose to use some of the feature reduction and selection techniques commonly used in data mining applications to reduce the computational and storage requirements of the intrusion detection methods. Since typically several of the user behavioral parameters are correlated, applying these techniques may reduce the number of parameters needed to represent the user behavior.

1 Introduction

With the ever increasing trend to make computer systems and databases easily accessible through the internet, there is also an increasing fear of intruders into the systems. One method of detecting intruders, who might enter the system in the guise of legitimate users is to maintain an audit of user activity [2, 3, 9]. Each audit record would summarize the user activity or a process activity in terms of a set of feature values. One of the main challenges in intrusion detection is to summarize the historical audit data so that when current activity is compared with it, one can classify it as legitimate or intrusive. Since the comparisons need to be made on-line, when the activity is taking place, it is necessary to minimize the processing time for such detections. This also puts a constraint on the size of the data we wish to keep in the summary.

Data mining techniques deal with very large volumes of data [14, 8]. In order for these techniques to be effective in on-line processing of queries, they attempt to reduce the data that they need to process in answering a query. These are referred to as data reduction techniques. Feature reduction is one of many such techniques [13, 14].

In this paper, we investigate the effectiveness of integrating some data reduction techniques of data mining with the intrusion detection techniques. The work presented in this paper is preliminary in nature. Thus the insights gained are explained more through examples rather than any formal proofs. As we continue to make progress in these efforts, we hope to obtain more concrete evidence.

In particular, we look at the significance tests on features that determine whether or not a feature contributes in correctly classifying an observed user activity as intrusive or non-intrusive. The activity itself is characterized in terms of a set of features in an audit record. We also look at measures of mutual significance. These measures are useful when all features in

an audit record are not completely independent. In fact, dependencies do exist among features in real systems. When the degree of mutual dependency is high, then we could possibly eliminate one or more of the features from the audit record without affecting the intrusion capabilities. We also look at logic methods that can infer classification rules from a given set of audit records. This scheme also reduces the amount of data that need to be maintained about historical behavior of users. We find that all these techniques are quite useful in data reduction in intrusion detection schemes.

The paper is organized as follows. In Section 2, we summarize the use of statistical methods in intrusion detection. In Section 3, we discuss the data reduction techniques available in data mining. We also discuss the classification methods. Section 4 discusses in detail three feature-reduction methods used in data mining—significance testing, mutual significance, and logic methods. Section 5 illustrates these techniques through examples. In Section 6, we analyze the effect of data reduction on intrusion detection capability of systems. We look at systems with independent features, linearly dependent features, and features with non-linear dependencies. Finally, Section 7 has a summary of current work and plans for future work.

2 Intrusion Detection

Detecting intruders in a computer system is vital to many organizations. This is especially relevant to commercial and military organizations since they maintain several key data on their computer systems. While preventing intruders by means of firewalls and authentication mechanisms is a necessary step, the organizations would also like to be alerted when suspicious activity is observed on the system.

NIDES, the Next-generation Intrusion Detection Expert System, is developed at SRI International for the purpose of detecting anomalous behavior in computer system [2, 3]. The system is based on statistics—the normal behavior of a user is represented by a set of parameters. Statistic are dynamically calculated based on the user's behavior. Whenever a user's behavior (expressed in terms of audit records) is found to be (statistically) significantly deviate from the normal statistic, an alarm is raised. In order to keep up with the changing behavior of a legitimate user, the system gives higher weight to the recent behavior and lower weight to the past. This is achieved by defining half-life for the statistic.

Since no single measure may characterize the behavior of a user, NIDES measures and maintains statistic on several measures. Some of the measures that were used in one of their applications, Safeguard, include user time, system time, number of open files, elapsed process time, integral of memory usage, number of page faults, etc. [2]. For each measure, a probability distribution of short-term (immediate past) and long-term behaviors was constructed.

The degree of difference between long-term profile and short-term profile is measured in terms of a Q-statistic. This statistic is maintained for each measure. Whenever a new audit record is received by the system, indicating the current activity, the NIDES system generates a vector of Q values. Here, large Q values indicate deviation from the expected behavior and hence are suspicious. The distribution of Q for each measure is maintained as a histogram of bins. In fact, when an audit record is received, first an S-statistic is computed for each feature and the sum of squares of the S-statistic. This is referred to as T^2-statistic. This statistic along with the probability distributions of the past is used to generate a probability of such occurrence. This in turn is used to raise alarms.

The computational and storage requirements in NIDES is directly proportional to the number of measures (or features) being monitored. For each measure, a histogram and several statistic are maintained. This information is updated either after every audit record or after a certain period of time (or after certain number of audit records). This processing is to be done sufficiently frequently so as to avoid false alarms as well as the risk of missing detection of legitimate intrusions. Several computations are also needed to determine whether or not an intrusion has occurred.

While it is perfectly valid that we need to characterize the behavior of a user in terms of several measures, there is also a danger of keeping track of to many measures. In addition, these measures would not be completely independent. For example, large number of page faults will also result in high I/O activity. This also means larger turnaround time, more number of context switches for the process, and higher system time.

In the current work, we look at the means to identify any correlations that exist among these measures (or features). When high correlations are observed between measures, some of the measures could be safely omitted. Whether or not such omissions will affect intrusion detection is also a topic of discussion in this paper.

In this work we also propose alternate representations for characterizing user behavior. We especially illustrate the use of logic methods in arriving at rule-based schemes for intrusion detection.

3 Data Mining Techniques

Data mining is the search for valuable information in large volumes of data [1, 14]. The data itself is characterized by several features (referred to as measures in intrusion detection). In other words, the data consists of records with information related to an entity expressed in terms of characteristics or features. For example, the credit information of a credit-card holder may indicate the holder's credit limit, average monthly charges, percentage of times the payment was defaulted, the average annual earnings, etc. One of the primary applications of data mining is to predict the unknown behavior of a potential customer based on the known historical behavior of other customers.

Linear regression is one of the traditional techniques used for prediction in data mining [4]. Here, given historical data of say two variables, assuming that one variable is linearly dependent on the other, the regression techniques determine a linear relationship between them. In other words, the entire historical data is now converted to a simple linear equation.

The decision trees are also a powerful model used in data mining [4, 14, 13]. They are produced by several techniques including classification and regression trees and Chi-squared automatic induction. These are particularly useful for classification. For this reason, we propose to use these models to classify audit records into non-intrusive and intrusive classes. A decision tree is expressed as a clear sequence of decision rules. The rules themselves can be expressed as logic statements, in a language such as SQL, and hence can be applied to new audit records easily.

Techniques such as neural nets and genetic algorithms are also used for classification and prediction in data mining [4, 14]. However, in the current work we do not explore the application of these techniques for intrusion detection.

In addition, one of the main objectives of data mining techniques is to reduce the amount of data that need to be processed when an on-line query is received. Several reasons are mentioned for the reduction of data. They include: the expected time for inducing a solution may be too long or the time to read the data from the database may be too large. Different parameters associated with a record of interest in the database are referred to as features. Some of the operations suggested for data reduction are: (i) reduce number of features, (ii) reduce the number of records to be processed, and (iii) reduce the number of values for a feature (also smoothing). In the proposed work, we concentrate on the first method—reduce the number of features to be examined.

4 Methods for feature reduction

One of the main objectives of this paper is to suggest methods to reduce data that needs to be maintained and processed for intrusion detection. One method of data reduction is reduction of the features or parameters collected in each audit record. Here, we discuss some feature reduction techniques and how they can be effective in the reduction.

The objective of feature reduction is to determine a subset of features which best represent the behavior of a user as represented by the audit records. Obviously the reduced subset should be sufficient to detect any intrusions in the system. In this section, we discuss three methods for feature reduction: significance test for an individual feature (features that are not significant may be eliminated), mutual significance—significance of a pair of features (when a feature is present and significant, the other one may be redundant, and hence eliminated), and logic methods or rule-based methods to identify user characteristics (those features not appearing in the rules may be eliminated).

Since the main focus of this paper is intrusion detection, the role of a feature in an audit record is to help classifying it as either intrusive or non-intrusive, hence we use this context in the following sections.

4.1 Significance Test for a Feature

One of the characteristics that a feature should satisfy is whether or not it can significantly help in distinguishing a non-intrusive audit record from an intrusive one. Several statistical tests of significance currently exist in literature [6, 10]. For example, if we would like to determine whether or not CPU time is a feature of significance, we select n_I audit records which are known to be intrusive and n_{NI} non-intrusive (or normal behavior) records. One test of statistical significance is as follows:

$$sig(CPUtime) = \frac{|\mu_I - \mu_{NI}|}{se} \quad (1)$$

$$se(I, NI) = \sqrt{\frac{\sigma^2(I)}{n_I} + \frac{\sigma^2(NI)}{n_{NI}}} \quad (2)$$

Here, se is the variance of the distribution of the sample means (the standard error of the sampling distribution), and σ^2 is the sample variance for each class. The μ's are the means of CPU times in the two classes. Since the difference in means will be approximately normally distributed for large sample sizes, independent of the distribution of the samples, we can check whether or not sig(CPUtime) is significant at a given

level of significance. We illustrate this with an example.

Suppose we selected 200 non-intrusive and 100 intrusive audit records. The mean CPU time for the non-intrusive records is 30 msec and 40 msec for the intrusive records. The standard deviation for non-intrusive records is 20 msec and 30 msec for intrusive class. So se(I,NI)= 3.31 and sig(CPUtime)=3.01. If we are testing this at the 0.05 level of significance (or 95% confidence), the critical value is 1.96. Since the computed sig(CPUtime) is larger than the critical value, we can conclude that CPUtime may be a significant feature in distinguishing an intrusive activity from a non-intrusive one. In fact, in this case the feature is even significant at a 0.0026 level or 99.74% confidence level.

In cases where the significance of the feature is lower than the critical value at a given confidence level, we can say that the feature is not statistically significant in the classification of an audit record. Hence, it can probably be eliminated.

4.2 Mutual significance

In the above test, each feature is analyzed independently and tested for significance. However, if the features are analyzed collectively, then we may obtain more information about the features. For example, from the individual significance tests, if we were to determine that both CPU time and I/O time are significant in detecting an intrusion. Then we select both the features. However, if the features are correlated to each other, then one of them will suffice for intrusion detection. One of the methods to detect mutual significance is as follows:

1. Let $\vec{\mu_I}$ and $\vec{\mu_{NI}}$ be the vector of feature means for the intrusive and non-intrusive sets of audit records, respectively. Let Cov_I and Cov_{NI} be the covariance matrices for the intrusive and non-intrusive sets of audit records, respectively.
2. Compute $Dist=(\vec{\mu_I}\text{-}\vec{\mu_{NI}})(Cov_I+Cov_{NI})^{-1}(\vec{\mu_I}\text{-}\vec{\mu_{NI}})^T$
3. Determine the smallest set of features that can result in largest value of $Dist$.

Since determining optimal feature set may be computationally expensive, heuristic procedures are suggested to achieve near optimality. In general, the best set of k independent features are the k features with the largest values of $(\mu_I - \mu_{NI})^2/(\sigma_I^2 + \sigma_{NI}^2)$ [14].

For example, if we also considered the I/O time as a feature and the statistics for this feature are as follows: The mean I/O time for the 200 non-intrusive records is 400 msec and 300 msec for the 100 intrusive records. The standard deviation for non-intrusive records is 200 msec and 150 msec for intrusive cases. So se(I,NI)= 20.61 and sig(IOtime)=4.85. If we are testing this at the 0.05 level of significance (or 95% confidence), the critical value is 1.96. Since the computed sig(IOtime) is larger than the critical value, we can conclude that IOtime may be a significant feature in distinguishing an intrusive activity from a non-intrusive one. In fact, in this case the feature is even significant at a 0.001 level or 99.99% confidence.

Now, the question is whether or not both CPUtime and IOtime need to be considered for classifying an audit record as intrusive or non-intrusive, or one of them will suffice. Suppose we also computed the correlation coefficients between the two features in the intrusion and non-intrusion cases, and they are found to be ρ_I=0.75 and ρ_{NI} = 0.6. So the covariance in these two cases is given by cov_I=0.75*30*150=3375. Similarly, cov_{NI}=0.6*20*200=2400. The covariance matrices are given by:

$$C_I = \begin{bmatrix} 900 & 3375 \\ 3375 & 22500 \end{bmatrix}$$

$$C_{NI} = \begin{bmatrix} 400 & 2400 \\ 2400 & 40000 \end{bmatrix}$$

From here, distance measure can be computed as follows:

$$\begin{aligned} Dist &= ([M_I - M_{NI}])(C_I + C_{NI})^{-1}(M_I - M_{NI})^T \\ &= ([10 \quad -250]) \left(\begin{bmatrix} 1300 & 5775 \\ 5775 & 62500 \end{bmatrix} \right)^{-1} \left(\begin{bmatrix} 10 \\ -250 \end{bmatrix} \right) \\ &= 1.10 \end{aligned}$$

This is comparatively a small number indicating that the features are statistically close. We need to perform the same analysis for other feature pairs and pick the ones with large distance.

4.3 Logic Methods

Given the training data, if we can convert the data into a set of decision rules or a decision tree, then we could use the resulting form in intrusion detection.

Consider the case of decision rules [12, 14]. The rules are represented as a set of IF-Then propositions. Each rule has a conditional part and a conclusion part. The conditional part is a boolean expression in a propositional form. The conclusion is an assignment of class or value.

Each decision tree is a conjunction of true-or-false terms. For example, if the system has observed that whenever CPU time is higher than 3 minutes and I/O

time is greater than 4 minutes, it is an intrusion, then it is represented as:

$$CPUTime > 3.0 \wedge IOTime > 4.0 \rightarrow \text{Intrusion} \quad (3)$$

In addition, if the system observed that any time CPU time is higher than 8 minutes or I/O time greater than 15 minutes there is an intrusion, we can add two more rules to the above set:

$$CPUTime > 8.0 \rightarrow \text{Intrusion} \quad (4)$$

$$IOTime > 15.0 \rightarrow \text{Intrusion} \quad (5)$$

Similarly, the system may also infer similar rules about the non-intrusive or normal cases. For example, if the training data were to indicate that in all cases when CPU time is less than 1 minute and I/O time is less than 2 minutes there is no intrusion, then we can add another rule:

$$CPUTime < 2 \wedge IOTime < 1 \rightarrow \text{No intrusion} \quad (6)$$

The size of rule set depends on the number of parameters and the complexity of their relationship in the presence or absence of intrusion activity.

The second form of logic method to represent the intrusive/non-intrusive behavior is decision-tree [4, 5, 12, 14]. Decision trees are binary trees where the nodes represent decisions. For example, since CPUTime > 8 represents a case of intrusion, the root node could be this decision. Here, the branch with the attribute "True" would lead to a leaf node of type "intrusion." The branch with the attribute "False" has several alternatives. One choice is to have a node with the decision IOTime > 15. We can thus build the decision tree.

The size of the tree is a measure of the complexity of this algorithm. To be more precise, the depth of the tree represents an exact measure of the maximum comparisons (at the nodes) that need to be made to reach the final decision of intrusion or non-intrusion activity.

Often, we may encounter situations where the data presented in the training data is weak. For example, when CPU time and I/O time are the only two features to characterize the activity and if we found a record with CPUTime=4 and IOTime=3 to be intrusive while another one with CPUTime=3.9 and IOTime=3.2 to be non-intrusive. Should these two cases form two nodes in the decision tree or two rules in the decision rules? Obviously, if we were to include every such case as a node or a rule, the tree (or the set of rules) will be quite large and hence larger computational complexity. Alternatively, we can treat the special as an anamoly and ignore it. For example, if 10 records suggest that values with CPU time between 4.0 and 6.0 and I/O time between 8 and 12 are non-intrusive, but one intrusive record was found with CPU time = 5.5 and I/O time = 10.5. Then, we can either include it as a separate node or simply ignore the latter record treating it as an anmoly. Or it is likely that the two selected parameters are not adequate in identifying the intrusive cases.

5 Application of Data Reduction Techniques in Intrusion Detection

In this section, we illustrate the efficacy of the logic methods through examples. For easy readabaility, we describe the application and its results as a step-by-step procedure.

1. Select the number of features. We selected the following five features (F_1-F_5) to represent user behavior. These features are a subset of those chosen by the NIDES system in their experimentation [2, 3].

   Feature 1 (F_1): User CPU Time

   Feature 2 (F_2): System CPU time

   Feature 3 (F_3): Character IO during the application execution

   Feature 4 (F_4): The integral of real memory usage over time

   Feature 5 (F_5): Number of pages read in from the disk

2. Select the rules representing the normal behavior (i.e., non-intrusive) behavior of the system. We choose the following for the illustration.

$$F_1 < 10 \quad (7)$$

$$F_2 < 5 \quad (8)$$

$$F_3 < 200 \quad (9)$$

$$F_4 < 30 \quad (10)$$

$$F_5 < 700 \quad (11)$$

$$10 < F_1 < 20 \wedge 10 < F_2 < 15 \quad (12)$$

   In a practical situation, these rules are not available. However, since we need to generate data representing user behavior, we use these rules. the rules will also be used later to verify the efficacy of the logic methods. An audit record that satisfies any of the above rules is said to be non-intrusive or genuine. An audit record that does

1:	$F_5 > 699.5$	false: 2	true: 3
2:	answer= 1		
3:	$F_3 > 199.5$	false: 4	true: 5
4:	answer= 1		
5:	$F_4 > 29.5$	false: 6	true: 7
6:	answer= 1		
7:	$F_2 > 4.5$	false: 8	true: 9
8:	answer= 1		
9:	$F_1 > 9.5$	false: 10	true: 11
10:	answer= 1		
11:	$F_2 > 10.5$	false: 12	true: 13
12:	answer= 2		
13:	$F_1 > 19.5$	false: 14	true: 15
14:	$F_1 > 10.5$	false: 16	true: 17
15:	answer= 2		
16:	answer= 2		
17:	answer= 1		

Table 1: Rules derived from 20,000 audit records

not satisfy any of the above six rules is said to be intrusive.

3. Set limits for the values of the features. We set the following limits.

$$0 < F_1 < 30 \quad (13)$$
$$0 < F_2 < 15 \quad (14)$$
$$0 < F_3 < 400 \quad (15)$$
$$0 < F_4 < 60 \quad (16)$$
$$0 < F_5 < 1000 \quad (17)$$

4. Generate audit records by assigning random values to each of the five features. The data generated follow the limits mentioned above(eqns. 13-17). We generated 20,000 audit records.

5. Classify each record as either intrusive or non-intrusive based on the rules selected above. Out of the 20,000 generated, 19,432 were found to be non-intrusive and the rest 568 were intrusive type.

6. Run a program which can look at the training data of 20,000 audit records (with classification) and arrive at either a decision tree or a set of rules for classification. We used the programs dtree and logic supplied in [14]. Table 1 summarizes the decision tree obtained.

1:	$F_5 > 699.5$	false: 2	true: 3
2:	answer= 1		
3:	$F_4 > 29.5$	false: 4	true: 5
4:	answer= 1		
5:	$F_2 > 4.5$	false: 6	true: 7
6:	answer= 1		
7:	answer= 2		

Table 2: Rules derived for the correlated data

The table s to be interpreted as follows. The first column is the rule number. Rule 1 says that if feature 5 (F_5) is greater than 699.5, then go to rule 3; otherwise go to rule 2. In fact, rule 2 says that answer=1 or that the audit record that is being analyzed corresponds to class 1. In our case, class 1 corresponds to non-intrusive audit records. In other words, when $F_5 \leq 699.5$, an audit record can be automatically classified as a non-intrusive class. This matches with our original data generation rules (eqn. 7-12). When feature 5 is greater than 699.5, we go to rule 3 which further compares if $F_3 > 199.5$.

Since the rules derived from the audit data through logic methods match those of data generation, we can conclude that these methods were quite effective in intrusion behavior characterization.

In the above example, we chose all independent values for the features. Suppose we were to choose some dependencies between the feature values. In particular, if $F_1 = 2 * F_2$+random(0,5) and $F_5 = 2.5 * F_3$+random(0,50), then clearly there is a correlation between the features. When data was generated with these values, we obtained 17861 non-intrusive records and 2139 intrusive type. When the logic methods were run on the data, we obtained the rules for classification shown in Table 2.

From Table 2, we notice that only features 2, 4, and 5 were considered in the decision process. So the system was able to infer the correlation between features 1 and 2 and hence eliminated feature 1. Similarly it inferred the correlation between features 3 and 5 and eliminated feature 3. Thus, we need only features 2, 4, and 5 to detect intrusion. Thus the data size is reduced. If we do the significance tests on these features, as described in Section 4.1, we get the results of Table 3.

From this table, it is clear that all features are significant. This, however, does not convey the corre-

Feature	Significance
F_1	39.2
F_2	40.2
F_3	130.4
F_4	72.5
F_5	130.6

Table 3: Significance of different features

	F_1	F_2	F_3	F_4	F_5
F_1	1.0	0.99	-0.11	-0.06	-0.11
F_2	0.99	1.0	-0.11	-0.06	-0.11
F_3	-0.11	-0.11	1.0	-0.14	0.999
F_4	-0.06	-0.06	-0.14	1.0	-0.14
F_5	-0.11	-0.11	0.999	-0.14	1.0

Table 4: Correlation coefficients for Non-intrusive audit data

lation information among the features. The mutual significance of the features is evident when we look at the correlation features, for example. The correlation matrix for the above data is given in Tables 4 and 5.

From the correlation data, not surprisingly, we can observe that features 1 and 2 are highly correlated. Similarly, features 3 and 5 are also highly correlated. Hence, we can reduce the five features of the audit record to just three features.

If we were to apply the distance metric (*dist*) discussed in section 4.2, we would arrive at the same conclusion.

6 Effect of Data Deduction on Intrusion Detection

While the data reduction techniques may help in reducing the size of the audit record data, the main concern is one of false alarms or missed intrusion detections. In other words, we need to address the issue the loss (if any) in intrusion detection capability of the system when we apply the data reduction techniques.

	F_1	F_2	F_3	F_4	F_5
F_1	1.0	0.97	-0.03	-0.01	-0.03
F_2	0.97	1.0	-0.03	-0.02	-0.03
F_3	-0.03	-0.03	1.0	-0.01	0.99
F_4	-0.01	-0.02	-0.01	1.0	-0.01
F_5	-0.03	-0.03	0.99	-0.01	1.0

Table 5: Correlation coefficients for Intrusive audit data

For example, consider the case with correlated features discussed above. Let us remove features 1 and 3 from the audit record. The next question we need to deal with is the set of rules for intrusion detection.

1. Consider the rule $F_3 < 200$. Since we determined that F_3 and F_5 are correlated, we need to find a relationship between them. If we were to run a linear regression for F_5 in terms of F_3, we find that $F_5 = 2.5 * F_3 + 25$. In other words, we need to translate the $F_3 < 200$ as $F_5 < 562.5$. However, since the $F_5 < 700$ rule already covers this, we can eliminate the F_3 rule completely.

2. Now consider the rule $F_1 < 10$. Once again using a linear regression, we can determine that $F_1 = 2 * F_2 + 2.5$. Hence, the rule may be rewritten in terms of F_2 as $F_2 < 3.75$. Once again, the original rule $F_2 < 5$ already covers this. So we can drop the rule.

3. Now, we need to consider the rule $10 < F_1 < 20 \wedge 10 < F_2 < 15$. Once again, using the linear regression relationship derived above ($F_1 = 2 * F_2 + 2.5$) we can rewrite it as $3.75 < F_2 < 8.75 \wedge 10 < F_2 < 15$. Since the two ranges for F_2 do not overlap, we can safely eliminate this rule also.

Thus, the new set of rules for intrusion detection are

$$F_2 < 5 \quad (18)$$

$$F_4 < 30 \quad (19)$$

$$F_5 < 700 \quad (20)$$

When we ran the audit records under these rules, we found that all 17861 non-intrusive records were correctly identified as non-intrusive class. The 2139 intrusive records were also classified as intrusive. In other words, there was no loss of information due to the reduction of features from 5 to 3.

However, we need to understand the importance of either rerunning the logic methods to determine the new rules or to modify the original rules by incorporating the relationships between the eliminated features and the existing features.

6.1 Multi-dependency relationships

Suppose, the relationships among the features was not one-to-one as in the above example, but involve multiple features. For example, if we have the following relationships $F_1 = 1.5 * F_2 + 0.5 * F_4 + \text{random}(0,5)$

1:	$F_3 > 199.5$	false: 2	true: 3
2:	answer= 1		
3:	$F_5 > 699$	false: 4	true: 5
4:	answer= 1		
5:	$F_2 > 4.5$	false: 6	true: 7
6:	answer= 1		
7:	answer= 2		

Table 6: Rules derived for the multiply correlated data

and $F_3 = 5*F_4+3*F_1+F_2+\text{random}(50)$. When audit records were generated under these conditions, with the original rules, we obtained 19665 non-intrusive type and 335 intrusive type records. If we were to eliminate two of the five features from the audit records, then the original rules also need to be changed accordingly. When we applied the logic methods on this data, we obtained the rules of Table 6.

Once again, when the data was validated with the new rules and features 1 and 4 removed from the audit records, we found that the system correctly identified all 19665 non-intrusive records as non-intrusive and the 335 records as intrusive. Once again, there was no loss of information due to the reduction in the number of features.

6.2 Non-linear dependencies

So far we had considered only linear relationships among the features. Suppose there were to be a non-linear relationship among the features. For example, let $F_5 = F_4 * F_2 + F_1 + random(100)$. When we generated audit data with this relation, we obtained 19845 non-intrusive records and 155 intrusive records. When logic methods were employed on the new data set, we obtained the rules in Table 7.

When we checked the input records with the above rules, the system classified 19745 of the 19845 records (99.5%) non-intrusive. However the remaining 0.5% non-intrusive were falsely classified as intrusive. This would correspond to false alarms. Out of the 155 intrusive records, 147 or 95.5% were correctly classified as intrusive. However, the remaining 4.5% were classified as non-intrusive. But the deterioration in performance is acceptable in most cases when the corresponding saving in processing and storage is taken into account.

From these examples, we can conclude that the logic methods are quite effective in reducing features and arriving at a set of rules based on audit record data. However, one needs to be cautious in applying the technique as it may lead to a small percentage of

1:	$F_5 > 699.5$	false: 2	true: 3
2:	answer= 1		
3:	$F_1 > 19.5$	false: 4	true: 5
4:	$F_3 > 232$	false: 6	true: 7
5:	$F_3 > 199.5$	false: 8	true: 9
6:	answer= 1		
7:	$F_1 > 10.5$	false: 10	true: 11
8:	answer= 1		
9:	answer= 2		
10:	$F_1 > 9.5$	false: 12	true: 13
11:	answer= 1		
12:	answer= 1		
13:	answer= 2		

Table 7: Rules derived for the non-linearly correlated data

incorrect classifications. This was especially so when non-linear correlation between features was present.

7 Conclusion

While the area of data mining is rich with techniques to reduce time for on-line processing of records, intrusion detection in computer system requires tools to reduce its time in detecting possible intrusions. In the current work, we attempted to illustrate some of the data reduction techniques that may be effectively used for intrusion detection without compromising security. We have especially focussed on statistical techniques to test individual significance and mutual significance, and logic based methods such as decision trees. The preliminary results are encouraging. In the absence of real-data, we used simulated data to show the efficacy of the techniques.

In the future work, we propose to test these methods on real audit record data. We also propose to test it on data when millions of data records are available. In addition, we propose to combine the techniques of intrusion detection techniques such as NIDES with those of the data mining to reduce the on-line processing time for intrusion detection.

References

[1] P. Adriaans and D. Zantinge, "Data Mining," Addison-Wesley, 1998.

[2] D. Anderson, et al, "SAFEGUARD Final Report: Detecting unusual program behavior using the NIDES Statistical Component," Final Report, Computer Science Laboratory, SRI International, Dec. 1993.

[3] D. Anderson, et al, "Detecting unusual program behavior using the statistical component of the Next-generation Intrusion Detection Expert System (NIDES)," Technical Report, SRI-CSL-95-06, Computer Science Laboratory, SRI International, May 1995, 77 pages.

[4] M. J. A. Berry and G. Linoff, "Data Mining Techniques," John Wiley & Sons Inc., 1997.

[5] L. Brieman and P. Smyth, "Applying classification algorithms in practice," *Statistics and Computing,* Vol. 7, No. 1, pp. 45-56, 1997.

[6] R. M. Henkel, "Tests of Significance," SAGE Publications, 1976.

[7] R. Jain, "The Art of Computer Systems performance Analysis," John Wiley & Sons Inc., 1991.

[8] M. James, "Classification Algorithms," John Wiley & Sons Inc., 1985.

[9] H.S. Javitz and A. Valdes, "The SRI IDES Statistical Anomaly Detector," *Proc. IEEE Symp. Security and Privacy,* Oakland, CA, May 1991, pp. 316-326.

[10] G. K. Kanji, "100 Statistical Tests," SAGE Publications, 1993.

[11] T.Y. Lin, T.H. Hinke, D.G. Marks, and B. Thuriasingham, "Security and data mining," *Database Security IX: Status and prospects,* Eds. D.L. Spooner, S. A. Demurjian, and J. E. Dobson, Chapman & Hall, 1996, pp. 391-399.

[12] S. Weiss and N. Indurkhya, "Rule-based machine learning methods for functional prediction," *Journal of Artificial Intelligence Research,* Vol. 3, pp. 383-403, 1995.

[13] S. Weiss and C. Kulikowski, "Computer Systems That Learn: Classification, and Prediction Methods from Statistics, Neural Nets, machine Learning and Expert Systems," Morgan Kaufmann, 1991.

[14] S. M. Weiss and N. Indurkhya, "Predictive Data Mining: A Practical Guide," Morgan Kaufman Publishers, 1998.

Role-Based Access Control Session

Monday, July 26, 1999

Chair: Sylvia Osborn
University of Western Ontario

RBAC on the Web by Secure Cookies

Joon S. Park, Ravi Sandhu, and SreeLatha Ghanta
The Laboratory for Information Security Technology
Information and Software Engineering Department
George Mason University
{jpark, sandhu, sghanta}@list.gmu.edu

ABSTRACT: Current approaches to access control on Web servers do not scale to enterprise-wide systems, since they are mostly based on individual users. Therefore, we were motivated by the need to manage and enforce the strong access control technology of RBAC in large-scale Web environments. RBAC is a successful technology that will be a central component of emerging enterprise security infrastructures. *Cookies* can be used to support RBAC on the Web, holding users' role information. However, it is insecure to store and transmit sensitive information in cookies. Cookies are stored and transmitted in clear text, which is readable and easily forged.

In this paper, we describe an implementation of Role-Based Access Control with role hierarchies on the Web by secure cookies. Since a user's role information is contained in a set of secure cookies and transmitted to the corresponding Web servers, these servers can trust the role information in the cookies after cookie-verification procedures and use it for role-based access control. In our implementation, we used CGI scripts and PGP (Pretty Good Privacy) to provide security services to secure cookies. The approach is transparent to users and applicable to existing Web servers and browsers.

KEYWORDS: Cookie, Role-Based Access Control (RBAC), WWW Security

1 Introduction

WWW is commonplace. Increased integration of Web, operating system, and database system technologies will lead to continued reliance on Web technology for enterprise computing. However, current approaches to access control on Web servers are mostly based on individual users; therefore, they do not scale to enterprise-wide systems.

A successful marriage of the Web and a strong and efficient access control technology has potential for considerable impact on and deployment of effective enterprise-wide security in large-scale systems. Role-based access control (RBAC) [San98] is a promising technology for managing and enforcing security in large-scale enterprise-wide systems, and it will be a central component of emerging enterprise security infrastructures. We were motivated by the need to manage and enforce the strong access control technology of RBAC in large-scale Web environments.

To support RBAC on the Web, we chose a relatively mature technology, cookies - widely used on the Web - and have extended it for our purpose. Cookies were invented

to maintain continuity and state on the Web [KM97, KM98]. Cookies contain strings of text characters encoding relevant information about the user. Cookies are sent to the user's memory via the browser while the user is visiting a cookie-using Web site, and are stored on the user's hard disk after the browser is closed. The Web server gets those cookies back and retrieves the user's information from the cookies when the user later returns to the same Web site or domain. The purpose of a cookie is to acquire information and use it in subsequent communications between the Web server and the browser without asking for the same information again. Often a Web server sets a cookie to hold a pointer, such as an identification number, as a user-specific primary key of the information database maintained in the server. Technically, it is not difficult to make a cookie carry relevant information. For instance, a merchant Web server could use a cookie containing the user's name and credit card number. This is convenient for users, since they do not have to read lengthy numbers from their cards and key these in for every transaction. However, it is not safe to store and transmit this sensitive information in cookies because cookies are insecure. Cookies are stored and transmitted in clear text, which is readable and easily forged. Therefore, we should render secure cookies to carry and store sensitive data in them.

We will provide secure cookies with three types of security services: authentication, integrity, and confidentiality. Authentication services verify the owner of the cookies. Integrity services protect cookies against the threat that the contents of the cookies might be changed by unauthorized modification. Finally, confidentiality services protect cookies against the values of the cookies being revealed to an unauthorized entity. Details for these techniques have varying degrees of security and convenience for users and system administrators[1].

In this paper, we will describe how we implemented RBAC with role hierarchy [FCK95, SCFY96] on the Web using the secure cookies. To provide security services to secure cookies, we used CGI scripts and the PGP (Pretty Good Privacy) package, which are already in widespread current use.

The rest of this paper is organized as follows. First, in Section 2, we describe the technologies most relevant to our work, such as RBAC, cookies, and PGP. In Section 3, we describe security threats to cookies, and how to design secure cookies to support RBAC on the Web. In Section 4, we describe how we actually implemented RBAC on the Web by secure cookies, using CGI scripts and PGP on an existing Web server. Section 5 gives our conclusions.

2 Related Technologies

2.1 Role-Based Access Control (RBAC)

Role-based access control (RBAC) [San98] has rapidly emerged in the 1990s as a promising technology for managing and enforcing security in large-scale enterprise-wide systems. The

[1]For secure communications on the Web, we may consider using other existing technologies, such as, SHTTP (Secure HTTP [RS98, SR98]) and SSL (Secure Socket Layer [WS96]). However, these technologies cannot solve the *stateless* problem of HTTP. Furthermore, none of these can prevent end-system threats to cookies.

basic notion of RBAC is that permissions are associated with roles, and users are assigned to appropriate roles. This greatly simplifies security management.

A role is a semantic construct forming the basis of access control policy. With RBAC, system administrators can create roles, grant permissions to those roles, and then assign users to the roles on the basis of their specific job responsibilities and policy. Therefore, role-permission relationships can be predefined, which makes it simple to assign users to the predefined roles. Without RBAC, it is difficult to determine what permissions have been authorized for what users.

RBAC is a promising alternative to traditional discretionary and mandatory access controls, and ensures that only authorized users are given access to certain data or resources. It also supports three well-known security policies: data abstraction, least-privilege assignment, and separation of duties.

2.2 Cookies

At present, there are many browsers that support cookies, including Netscape, MS Internet Explorer, GNNWorks, NetCruiser and OmniWeb. Cookies have been used for many purposes on the Web, such as selecting display mode (e.g., frames or text only), shopping cart selections, and carrying names, passwords, account numbers, or some other bits of identifying data on the user's machine. Although there are many ways to use cookies on the Web, the basic process and the contents of cookies are similar. The detailed cookie specifications are available in [KM97, KM98].

Whenever a browser sends an HTTP request for a URL to a Web server, only those cookies relevant to that server will be sent by the browser. If the server finds any cookies that are related to the server, those cookies are used during this communication between the browser and the server. However, if the server does not find any cookies specified for it, either that server does not use cookies in the communication or the server creates new cookies for subsequent communication between the browser and the server.

Web servers may update the contents of their cookies for any specific circumstance. The cookie-issuer is not important for cookie validation. In other words, a server can create cookies for other servers in the domain. This is an important aspect of cookies that will be used in our implementations described in Section 4.

2.3 Pretty Good Privacy (PGP)

PGP (Pretty Good Privacy), a popular software package originally developed by Phil Zimmermann, is widely used by the Internet community to provide cryptographic routines for e-mail, file transfer, and file storage applications [Zim95]. A proposed Internet standard has been developed [CDFT98], specifying use of PGP. It uses existing cryptographic algorithms and protocols and runs on multiple platforms. It provides data encryption and digital signature functions for basic message protection services.

PGP is based on public-key cryptography. It defines its own public-key pair management system and public-key certificates. The PGP key management system is based on the relationship between key owners, rather than on a single infrastructure such as X.509.

Basically, it uses RSA for the convenience of the public-key cryptosystem, message digests (MD5, IDEA) for the speed of process, and Diffie-Hellman for key exchange. The updated version supports more cryptographic algorithms.

Even though the original purpose of PGP is to protect casual e-mail between Internet users, we decided to use the PGP package. The package is already widely used and satisfies our requirements, in conjunction with Web servers via CGI scripts for our implementation to protect cookies. These cookies have role information of the user.

3 Secure Cookies

3.1 Security Threats to Cookies

We distinguish three types of threats to cookies: *network security threats*, *end-system threats* and *cookie-harvesting threats*. Cookies transmitted in clear text on the network are susceptible to snooping (for subsequent replay) and to modification by network threats. Network threats can be foiled by use of the Secure Sockets Layer (SSL) protocol [WS96] which is widely deployed in servers and browsers.[2] However, SSL can only secure cookies while they are on the network. Once the cookie is in the browser's end system it resides on the hard disk or memory in clear text. Such cookies can be trivially altered and can be easily copied from one computer to another, with or without connivance of the user on whose machine the cookie was originally stored. We call this the end-system threat. The ability to alter cookies allows users to forge authorization information in cookies and to impersonate other users. The ability to copy cookies makes such forgery and impersonation all the easier. Additionally, if an attacker collects cookies by impersonating a site that accepts cookies from the users (who believe that they are communicating with a legitimate Web server), later he can use those harvested cookies for all other sites that accept those cookies. We call this the cookie-harvesting threat. These attacks are all relatively easy to carry out and certainly do not require great hacker expertise.

3.2 Designing Secure Cookies for RBAC on the Web

In this subsection, we describe how to transform regular cookies - which have zero security - into secure cookies, which provide the classic security services against the three types of threats to cookies (described in the previous subsection).

Secure cookies provide three types of security services: *authentication*, *integrity*, and *confidentiality services*. Selection of the kinds and contents of secure cookies depends on applications and a given situation. However, at least one authentication cookie and the Seal_Cookie - which provides the integrity service to the cookies - must be used with other cookies to frame basic security services, regardless of applications.

Figure 1 shows a set of secure cookies that we will create and use for RBAC on the Web. The Name_Cookie contains the user's name (e.g., Alice), and the Role_Cookie holds

[2]In many cases due to export restrictions from USA only weak keys (40 bits) are supported, but SSL technology is intrinsically capable of very strong protection against network threats.

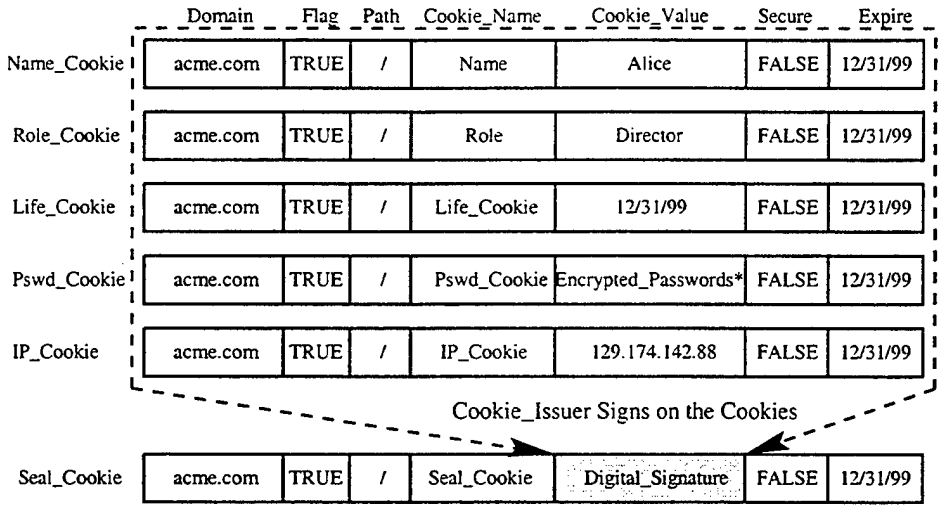


Figure 1: A Set of Secure Cookies for RBAC on the Web

the user's role information (e.g., Director). The Life_Cookie is used to hold the lifetime of the secure-cookie set in its Cookie_Value field and enables the Web server to check the integrity of the lifetime of the secure-cookie set. To protect these cookies from possible attacks, we will use IP_Cookie, Pswd_Cookie, and Seal_Cookie. Authentication cookies (i.e., IP_Cookie and Pswd_Cookie) verify the owner of the cookies by comparing the authentication information in the cookies to those coming from the users. The IP_Cookie holds the IP number of the user's machine, and the Pswd_Cookie holds the user's encrypted passwords. This confidentiality service protects the values of the cookies from being revealed to unauthorized entity. In our implementation, we used the IP_Cookie and Pswd_Cookie together to show the feasibility, but only one of those authentication cookies can be used to provide the authentication service. The choice of an authentication cookie depends on the situation.[3] Finally, the Seal_Cookie - which has the digital signature of the cookie-issuing server on the secure cookie set - supports integrity service, protecting cookies against the threat that the contents of the cookies might be changed by unauthorized modification.

There are basically two cryptographic technologies applicable for secure cookies: public-key-based and secret-key-based solutions. In our implementation, we use the public-key-based solution for security services provided by a PGP package via CGI scripts. In the next section, we will describe secure cookie creation, verification, and use of the role information in the Role_Cookie for RBAC with role hierarchies, in turn.

[3]It is also possible for authentication to be based on use of RADIUS [RRSW97], Kerberos [SNS88, Neu94], and similar protocols. Our focus in this work is on techniques that make secure cookies self-sufficient rather than partly relying on other security protocols, which is always possible.

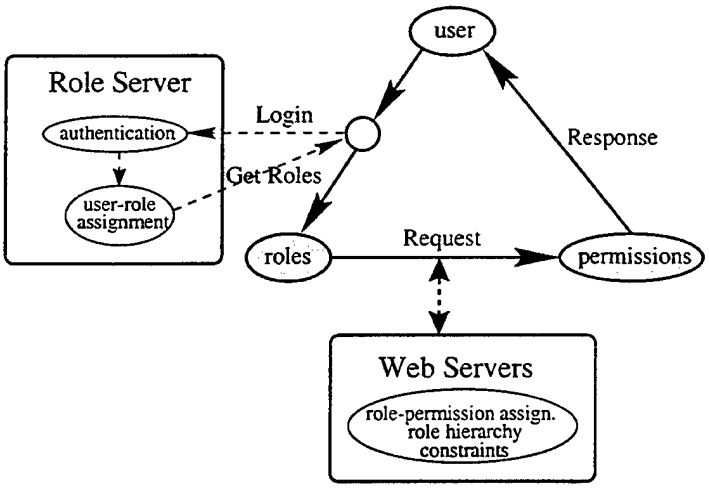


Figure 2: A Schematic of RBAC on the Web

4 RBAC Implementation by Secure Cookies

Figure 2 shows a schematic of RBAC on the Web. The role server has user-role assignment information for the domain. After a successful user authentication, the user receives his or her assigned roles in the domain from the role server. Later, when the user requests access to a Web server with the assigned roles in the domain, the Web server allows the user to execute transactions based on the user's roles instead of identity. The Web servers may have role hierarchies or constraints based on their policies.

However, how can the Web servers trust the role information presented by users? For instance, a malicious user may have unauthorized access to the Web servers by using forged role information. Therefore, we must protect the role information from being forged by any possible attacks on the Web as well as in the end-systems.

There can be many possible ways to support the above requirement. In this paper, as one possible solution, we will describe how to protect the role information from possible threats using secure cookies, and how we implemented RBAC (Role-Based Access Control) with role hierarchy on the Web. Figure 3 shows how the secure cookies (including a Role_Cookie) for RBAC are created and used on the Web. If a user, let's say Alice, wants to execute transactions in the Web servers in an RBAC-compliant domain, she first connects to the role server in the beginning of the session. After the role server authenticates Alice, it finds Alice's explicitly assigned roles in the URA (User-Role Assignment [SP98, SB97]) database and creates a set of secure cookies: Name_Cookie, Life_Cookie, Role_Cookies, IP_Cookie, Pswd_Cookie, and Seal_Cookie. Then, those secure cookies are sent to and stored in Alice's hard drive securely so that Alice does not need to go back to the role server to get her assigned roles until the cookies expire. Namely, she can use the roles in her Role_Cookie securely in the RBAC-compliant domain as long as the cookies are valid.

When Alice requests access to a Web server - which has PRA (Permission-Role As-

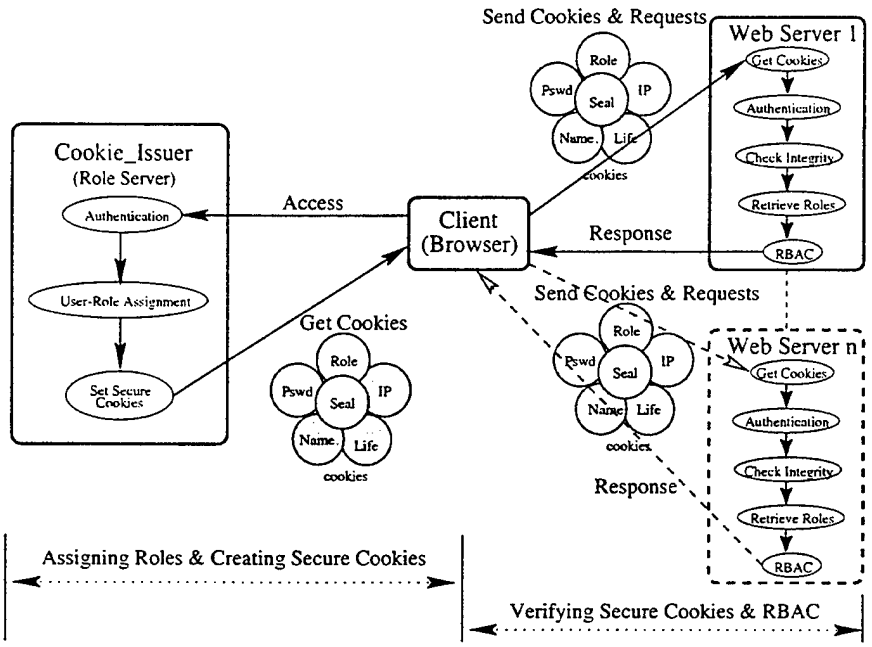


Figure 3: RBAC on the Web by Secure Cookies

signment [SBC+97]) information - by typing the server URL in her browser, the browser sends the corresponding set of secure cookies to the Web server: Name_Cookie, Life_Cookie, Role_Cookies, IP_Cookie, Pswd_Cookie, and Seal_Cookie. The Web server authenticates the owner of the cookies by using the IP_Cookie and Pswd_Cookie, comparing the value in the cookies with the values coming from the user. The user's passwords are encrypted in the Pswd_Cookie using the Web server's public key. The Web server decrypts the value of the Pswd_Cookie by using the corresponding key to read the user's passwords. Finally, the Web server checks the integrity of the cookies by verifying role server's digital signature in the Seal_Cookie using the role server's public key. If all the cookies are valid and verified successfully, the Web server trusts the role information in the Role_Cookie and uses it for RBAC with a role hierarchy and permission-role assignment information in the Web server.

4.1 Secure Cookie Creation

When a user, Alice, connects to the role server (which supports HTTP) of the domain with her Web browser, she is prompted by the HTML form to type in her user ID and passwords for the domain. We used the POST[4] method to send the information to the role server and the ACTION field to specify our Cookie-Set CGI program (set-cookie.cgi), to which the form data is passed. Figure 4 is a collaborational diagram in UML (Unified Modeling

[4]The GET request is very similar to the POST except that the values of the form variable are sent as part of the URL. However, the POST method sends the data after all their request headers have been sent to the server.

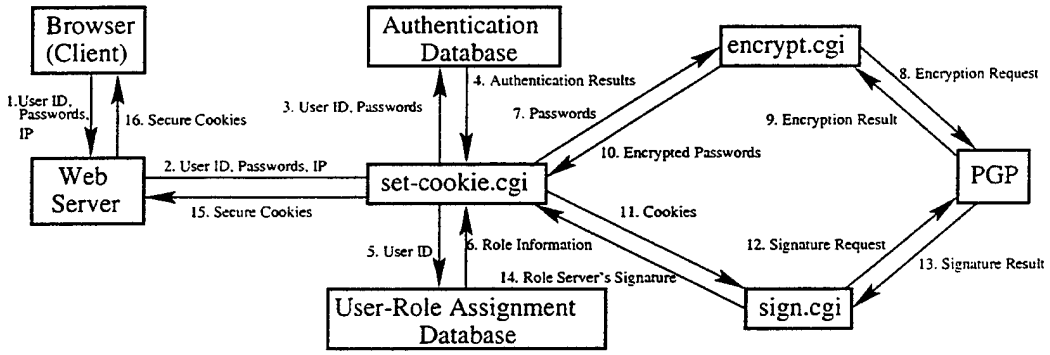


Figure 4: Creating Secure Cookies

Language [BJR98]) style notation for secure cookie creation. This diagram shows how we create a set of secure cookies for our implementation (refer to the left side of Figure 3).

The Web server receives the request headers, which include the address to the "set-cookie.cgi" program on the server. The server translates the headers into environment variables and executes the program. The "set-cookie.cgi" program first retrieves the user ID and passwords, and the IP number of the client machine from the environment variable, REMOTE_ADDR. The program authenticates the user by comparing the user ID and passwords with the ones in the authentication database.[5] It then assigns the user to roles by matching the user ID and the corresponding roles from the URA (User-Role Assignment) database.

Subsequently, a subroutine for encryption is called to another CGI program (encrypt.cgi), which uses PGP to encrypt the passwords by the cookie-verifying Web server's public key. These encrypted passwords will be stored in the Pswd_Cookie by the "set-cookie.cgi" program. Then, the "set-cookie.cgi" program creates IP_Cookie, Pswd_Cookie, Name_Cookie, Life_Cookie, and Role_Cookie, giving each cookie the corresponding value: IP number of the client machine, encrypted passwords, user's name, lifetime of the cookie set, and assigned roles.

To support the integrity service of the cookies, the "set-cookie.cgi" program calls another CGI program (sign.cgi), which uses PGP to sign on the message digest with the role server's private key. The "set-cookie.cgi" then creates the Seal_Cookie, which includes the digital signature of the role server on the message digest of the cookies.

Finally, the Web server sends the HTTP response header, along with the cookies, back to the user's browser, and the cookies are stored in the browser until they expire. These secure cookies will be verified and used in the Web servers as described in the following subsections. Figure 5 is an actual snapshot of a set of secure cookies from our implementation that are stored in the user's machine after the cookies are generated by the cookie-issuing Web server. The contents of the cookies exactly reflect the ones presented in Figure 1. Each

[5]If the user already has an authentication cookie in a set of secure cookies, Web servers can use the authentication cookie for user authentication instead of authentication databases.

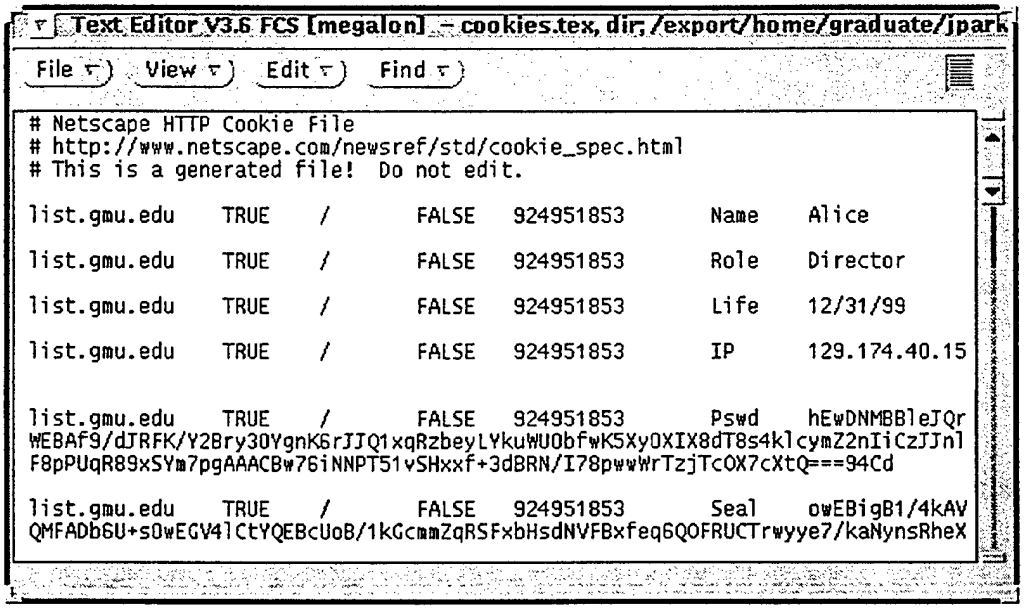


Figure 5: An Example of Secure Cookies Stored in a User's Machine

cookie has its corresponding domain, flag, path, security flag, expiration date, name, and value. The user's name, role, lifetime of the cookie set, IP number, encrypted passwords, and the digital signature of the cookie-issuing Web server on the cookies are stored in the corresponding cookies.

4.2 Secure Cookie Verification

Figure 6 is a collaborational diagram in UML style notation for secure cookie verification. This diagram shows how we verify (corresponding to the right side of Figure 3) the set of secure cookies that we generated in the previous subsection for our implementation. When Alice connects to a Web server (which accepts the secure cookies) in an RBAC-compliant domain, the connection is redirected to the "index.cgi" program. The related secure cookies are sent to the Web server and she is prompted by the HTML form to type in her user ID and passwords. The "index.cgi" program first uses the HTTP_COOKIE environment variable to retrieve the secure cookies (Name_Cookie, Life_Cookie, Role_Cookies, IP_Cookie, Pswd_Cookie, and Seal_Cookie) for the Web server. It then checks the validity of all the cookies. The two IP addresses, one from the IP cookie and the other from the environment variable, REMOTE_ADDR, are compared. If they are identical, then the host-based authentication is passed, and a hidden field "status" with the value of "IP-passed" is created to indicate that this stage was passed[6]. However, if the IP numbers are different, the user

[6]We used a hidden field to check the completion of the previous stage, which is passed on to the next program. This hidden field protects the pages from being accessed directly, skipping required verification steps, by a malicious user. For example, without this hidden field, a malicious user can access the pages

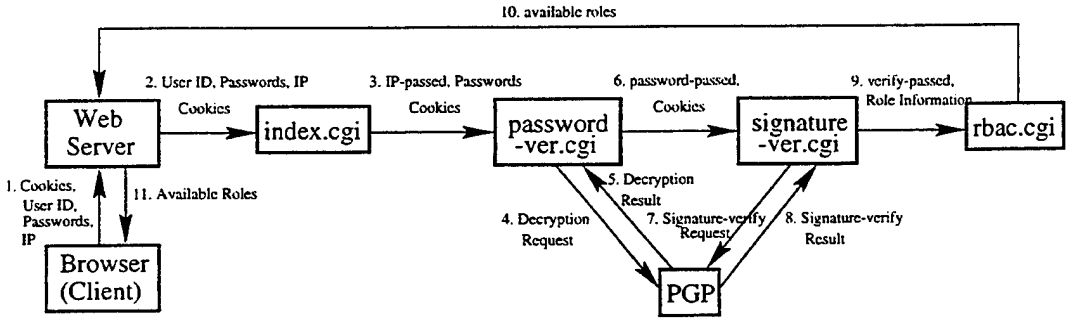


Figure 6: Verifying Secure Cookies

is rejected by the server.

When the user submits her user ID and passwords to the server, the Web server translates the request headers into environment variables, and another CGI program, "password-ver.cgi," is executed. We used the POST method to send the information to the Web server and the ACTION field to specify the CGI program (password-ver.cgi) to which the form data are passed. The first thing the "password-ver.cgi" does is to check the hidden field "status" to see if the previous stage was successfully completed. If this is "IP-passed," the program decrypts the value of the Pswd_Cookie (encrypted user password) using the PGP with the Web server's private key, since it was encrypted with the Web server's public key by the role server. The program (password-ver.cgi) then compares the two passwords: one from the user and the other decrypted from the Pswd_Cookie. If they are identical, then the user-based authentication is passed, and a hidden field "status" with the value of "password-passed" is created to indicate that this stage was passed. However, if the two passwords are different, the user has to start again by either retyping the passwords or receiving new cookies from the role server.

After the password verification is completed, another CGI program, "signature-ver.cgi," is activated to check the integrity of the cookies. Like the other programs, it first checks the value of "status" passed on from the previous program, and it proceeds only if it shows the user has been through the password verification stage. If the value is "password-passed," then the program verifies the signature in the Seal_Cookie with the role server's public key using PGP. If the integrity is verified, it means that the cookies have not been altered, and a hidden field "status" with the value of "verify-passed" is created to indicate that this stage was passed and forwarded to the final program, "rbac.cgi." This program uses the role information in the Role_Cookie for role-based access control in the server as described in the following subsection. However, if the signature verification is failed, the user has to start again by receiving new cookies from the role server.

directly with forged cookies.

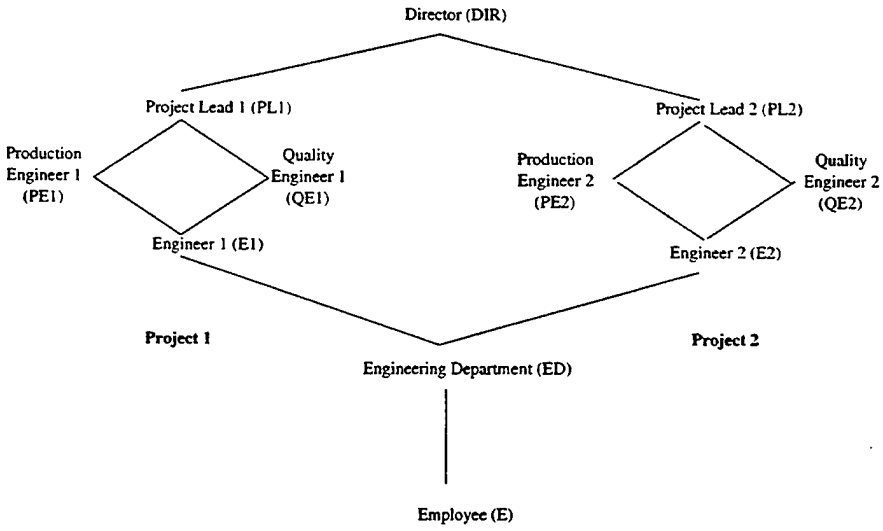


Figure 7: An Example Role Hierarchy

4.3 RBAC in the Web Server

After verifying all the secure cookies, the Web server allows the user, Alice, to execute transactions based on her roles, contained in the Role_Cookie, instead of her identity. In other words, the Web server does not care about the user's identity for authorization purposes. This resolves the scalability problem of the identity-based access control, which is being used mostly in existing Web servers. Furthermore, the Web server can also use a role hierarchy, which supports a natural means for structuring roles to reflect an organization's lines of authority and responsibility. Each Web server may have a role hierarchy defferent from that in other servers. In our implementation, we used a role hierarchy in the Web server, depicted in Figure 7. The location of RBAC-compliant Web servers is geographically free from that of the role server, since cookies can be issued by one Web server for use by others, regardless of their physical location.

If the "rbac.cgi" program in Figure 6 receives the value, "verify-passed," from the previous verification step, it means that the cookies have successfully passed all the verification stages, such as IP, passwords, and signature verification. Therefore, the Web server can trust the role information in the Role_Cookie, and uses it for role-based access control in the server.

Suppose Alice has the role DIR[7] in her Role_Cookie and she wants to access resources for PE1 - which require the PE1 role or roles senior to the PE1 role in the role hierarchy - in a Web server. First, she has to prove that the cookies she is presenting are genuine. To prove this, she has to go through all the verification steps: IP, passwords, and signature verification. She cannot jump ahead or skip any verification stage, as each program requires a hidden field, "status," from the previous stage. After the Web server has successfully

[7]Multiple roles can be stored in a Role_Cookie.

completed all the verification steps, the "rbac.cgi" program retrieves the role information, DIR, from Alice's Role_Cookie, and shows all the available roles[8] based on the role hierarchy depicted in Figure 7. Since she wants access to the pages that require PE1's privilege, she chooses the PE1 role to activate it among her available roles. Then she has the permissions assigned to the PE1 role and the roles junior to PE1, such as E1, ED and E. Now, when Alice requests access to a particular page in the server, the page checks if her activated role, PE1, has permission to access the page.

The user can use any roles among her available roles by activating them. For instance, if Alice would activate PL1, then she would be allowed to access the pages, which require the PE1 role, since PL1 is senior to PE1 in the role hierarchy. However, if she were to activate E1, then she would not be allowed to access the pages, since E1 is junior to PE1. This supports least privileges, since only those permissions required for the tasks are turned on by activating the required role.

How then can the Web server protect the pages from being accessed by unauthorized users? Suppose a malicious user, Bob, has the role PE1 but wishes to access pages that require the PL1 role. He could change the value of his Role_Cookie so that it has PL1, or roles senior to PL1. He would go through the password verification stages, since he would be able to log in as Bob by using his own passwords. However, when his Seal_Cookie is being verified, there would be a problem, as the signature verification would fail. Therefore, he would not be allowed to move beyond this stage. On the other hand, he could try accessing the pages directly by typing the URLs. This would not be allowed, since each page checks to see if he has activated the required role, PL1, or roles senior to PL1. In other words, Bob is not allowed to access the pages, which require roles senior to his, because he cannot activate the senior roles, which are out of his available role range.

As a result, the Web server allows only users, who have gone through all the verification steps with the secure cookies (Name_Cookie, Life_Cookie, Role_Cookies, IP_Cookie, Pswd_Cookie, Seal_Cookie), to access the pages. This access also is possible only if the users have the required roles and activate them among their available roles based on the role hierarchy.

5 Conclusions

In this paper, we have described how we implemented RBAC with role hierarchies on the Web using *secure cookies*. To protect the role information in the cookies, we provided security services, such as authentication, confidentiality, and integrity, to the cookies using PGP and CGI scripts in the Web servers. The cookie-issuing Web server creates a set of secure cookies including the user's role information, and other Web servers use the role information for RBAC with role hierarchies after cookie verification. This access control mechanism solves the scalability problem of existing Web servers. The use of secure cookies is a transparent process to users and applicable to existing Web servers and browsers.

[8] In this example, all the roles from E to DIR are available to Alice, since she has the senior most role in the role hierarchy.

6 Acknowledgements

This work is partially supported by the National Security Agency under its IDEA program.

References

[BJR98] Grady Booch, Ivar Jacobson, and James Rumbaugh. *The unified modeling language user guide.* Addison-Wesley, 1998.

[CDFT98] J. Callas, L. Donnerhacke, H. Finney, and R. Thayer. *OpenPGP massage Format*, November 1998. RFC 2440.

[FCK95] David Ferraiolo, Janet Cugini, and Richard Kuhn. Role-based access control (RBAC): Features and motivations. In *Proceedings of 11th Annual Computer Security Application Conference*, pages 241–48, New Orleans, LA, December 11-15 1995.

[Her96] Eric Herrmann. *CGI Programming with Perl 5.* Sams Net, 1996.

[KM97] David M. Kristol and Lou Montulli. *HTTP state management mechanism*, February 1997. RFC 2109.

[KM98] David M. Kristol and Lou Montulli. *HTTP state management mechanism*, July 1998. draft-ietf-http-state-man-mec-10.txt.

[Neu94] B. Clifford Neuman. Using Kerberos for authentication on computer networks. *IEEE Communications*, 32(9), 1994.

[RRSW97] C. Rigney, A. Rubens, W. A. Simpson, and S. Willens. *Remote Authentication Dial In User Service RADIUS*, April 1997. RFC 2138.

[RS98] E. Rescorla and A. Schiffman. *Security Extensions For HTML*, June 1998. draft-ietf-wts-shtml-05.txt.

[San98] Ravi Sandhu. Role-based access control. *Advances in Computers*, 46, 1998.

[SB97] Ravi Sandhu and Venkata Bhamidipati. The URA97 model for role-based administration of user-role assignment. In T. Y. Lin and Xiaolei Qian, editors, *Database Security XI: Status and Prospects.* North-Holland, 1997.

[SBC+97] Ravi Sandhu, Venkata Bhamidipati, Edward Coyne, Srinivas Ganta, and Charles Youman. The ARBAC97 model for role-based administration of roles: Preliminary description and outline. In *Proceedings of 2nd ACM Workshop on Role-Based Access Control*, pages 41–50. ACM, Fairfax, VA, November 6-7 1997.

[SCFY96] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, February 1996.

[SNS88] J.F. Steiner, C. Neuman, and J.I. Schiller. Kerberos: An authentication service for open network systems. In *Proc. Winter USENIX Conference*, 1988.

[SP98] Ravi Sandhu and Joon S. Park. Decentralized user-role assignment for Web-based intranets. In *Proceedings of 3rd ACM Workshop on Role-Based Access Control*, pages 1–12. ACM, Fairfax, VA, October 22-23 1998.

[SR98] A. Schiffman and E. Rescorla. *The Secure HyperText Transfer Protocol*, June 1998. draft-ietf-wts-shttp-06.txt.

[WS96] D. Wagner and B. Schneier. Analysis of the SSL 3.0 protocol. In *Proceedings of the Second UNIX Workshop on Electronic Commerce*, November 1996.

[Zim95] Phillip R. Zimmermann. *The Official PGP User's Guide.* MIT Press, 1995.

eMEDAC: Role-based Access Control Supporting Discretionary and Mandatory Features

Ioannis Mavridis, George Pangalos
Informatics Lab., General Dept., Faculty of Technology, Aristotle University of Thessaloniki, 54006, Greece
Email: imav@eng.auth.gr, gip@eng.auth.gr

Marie Khair
Department of Computer Science, Faculty of Natural and Applied Sciences, Notre Dame University, Louaize, Lebanon
Email: mkhair@ndu.edu.lb

Abstract

In this paper, we present an enhanced use of RBAC features in articulating a security policy for access control in medical database systems. The main advantage of this implementation is that it supports both MAC and DAC features at the same time; a feature that has been proved to be necessary in healthcare environments. The eMEDAC security policy that results from the above implementation provides an enhanced redefinition of a number of mechanisms of the already known MEDAC security policy. The concept of hyper node hierarchies is proposed for deriving totally ordered security levels while preserving the role hierarchy levels required satisfying particular administration needs. Finally, a demonstration example is given based on the pilot implementation of the proposed security policy in a major Greek hospital. The advantages offered are related to the efficiency of access control, the flexibility and decentralisation of administration, and the storage savings.

1. Introduction

Until now, in the key area of access control three major categories of security policies have been proposed that are usually used in computer systems: the discretionary policies, the mandatory policies, and the role-based policies. A recent well known role-based approach is RBAC (Role Based Access Control), which has received considerable attention as a promising way to enhance traditional discretionary (DAC) and mandatory (MAC) access controls.

According to [16], an important characteristic of RBAC is that by itself it is policy neutral. Furthermore, the security policy enforced in a particular system could be the net result of the appropriate configuration and interactions of various RBAC components (mechanisms). Another important benefit of RBAC is the ability to modify the security policy to meet the changing needs of an organization. However, most of the research work today, has being focused on the support of the DAC philosophy using the RBAC approach ([4], [5], [14], [21]).

The scope of this paper is to present an exploitation of RBAC features in articulating a security policy for access control in medical database systems (where both mandatory and discretionary features are needed). To achieve this, the mechanisms of the already known MEDAC security policy have been redefined on the basis of RBAC components. The resulting security policy (eMEDAC) is an enhancement of MEDAC that offers significant advantages, especially in access control, administration and storage requirements (with some drawbacks of additional computational workload).

2. The MEDAC Security Policy

Security policies are sets of principles and high level guidelines that concern the design and management of access control systems ([2]). Mechanisms are low level software and hardware functions, which can be configured to implement a security policy ([15]). In general, there are no policies that are better than others. This is because, not all systems have the same protection requirements. Policies suitable for a given system may not be suitable for another.

The choice of security policy depends on the particular characteristics of the environment to be protected. However. in recent years there is an increasing consensus that there are legitimate policies that have aspects of both mandatory and discretionary security models. Role-based policies are an example of this fact.

In health care environments there is a need for a security policy that is able to satisfy all the security components (availability, integrity and confidentiality). As has been demonstrated previously ([7], [12], [13]), both DAC and MAC when used separately have their limitations in achieving this. As a result, we have previously proposed a security policy called MEDAC (MEdical Database Access Control) based on both MAC and DAC approaches, that has been proved to be able to satisfy all the needed security requirements.

2.1. Overview of the Original MEDAC Security Policy

The MEDAC security policy is based on the Bell-LaPadula model ([2]) and takes advantage by utilizing the characteristics of discretionary, mandatory and role-based approaches. A detailed presentation of MEDAC can be found in [7], [11], [12], [13]. It is based on the following principles:

- Every authorized person has the right to access the system. For this reason, each user accessing directly the system must have an account allowing him to log into the system. Every account is associated to a predefined user role. This user role represents the user task in the application.
- Every user role has a clearance level and a category set. The category depends on the nature of the user role and the data sets it needs to access. The clearance level of the user role represents its trustworthiness.
- Every data set is assigned a sensitivity level and a category set. The sensitivity level reflects its sensitivity depending on its context, content, exterior factors or specific situations (e.g. related to sensitive cases). The category depends on the use and the nature of the data set.

The levels within a certain category are completely ordered (hierarchical), while the categories are partially ordered.

The MEDAC security policy is formed of three layers that every user requesting to access the database has to pass through. The MEDAC model is shown schematically in figure 1. More specifically:

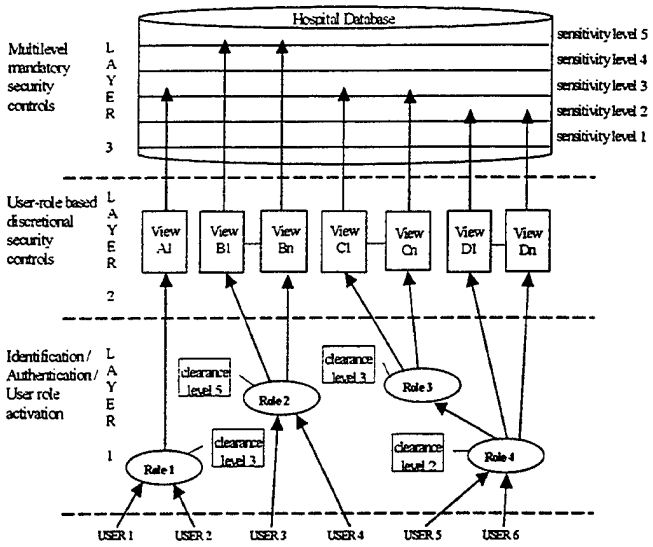


Figure 1. The MEDAC model.

- **First layer:** In this layer, the identification and authentication of a particular user is performed by the security mechanisms of the host system. After his identification, the user activates a user role from the set of roles that has been assigned to him initially.
- **Second layer:** User roles in this layer are permitted to access the database via a discretional access control. Each user role has the authorisation to see a certain view and to exercise just the authorised access rights. The view has been predefined depending on the need-to-know requirements of the users performing this role.
- **Third layer:** Users roles are required to access the database satisfying the multilevel access control. Depending on the clearance level of the specific user role, it will be able to access just the part of the database which is defined according to the dominance relation (i.e., the sensitivity level of the information which can be read by the user role is less or equal to its clearance level).

MEDAC has been implemented already in the AHEPA general hospital of Thessaloniki. This experimental implementation has produced successful results ([12], [13]).

2.2. Supported Characteristics of the Major Security Models

- 'DAC' Characteristics: Discretionary controls are supported in the MEDAC policy as a combination of the access-matrix mechanism and specialized views that are used to define the sets of permitted types of access that can be executed by specific user roles to data sets. Schematically, the DAC aspects of the MEDAC policy are handled in the 2nd layer of the MEDAC model (Figure 1).
- 'MAC' Characteristics: In the MEDAC policy the multilevel model of MAC is supported. An important characteristic of the MEDAC policy is that mandatory security controls are used additionally to the discretional security ones. Mandatory access control is based on the sensitivity of the personal and medical data of patients in order to ensure their protection in a strict way ([9], [13]). Furthermore, the use of multilevel security has made it possible to provide generalized explanations (cover stories) for unavoidable observable information that would otherwise lead to partial or complete inference of sensitive information ([10]). Schematically, the MAC aspects of the MEDAC policy are handled in the 3rd layer of the MEDAC model.
- 'RBAC' Characteristics: MEDAC also partly supports the RBAC approach, mainly by means of user roles and the user role hierarchy. The use of the concept of user roles has made it possible for every user to have access just to that part of the application he needs to use in order to perform his specific task ([10]). Schematically, the RBAC aspects of the MEDAC policy are handled in the 1st layer of the MEDAC model (Figure 1). However, although MEDAC takes into account the RBAC concept, it does not utilize the RBAC features fully, since it has been developed independently, before RBAC was widely accepted.

3. The Enhanced MEDAC Security Policy (eMEDAC)

3.1. The RBAC Components Used

Role-based policies allow the specification of authorizations to be granted to users on objects like in the discretionary approach, together with the possibility of specifying restrictions on the assignment or on the use of such authorizations. They also provide a classification of users according to the activities they execute. Analogously, a classification is recommended for objects according to their type or to their application area ([15]).

RBAC is an emerging role-based policy ([6], [17], [18]). Role hierarchies are used in RBAC for structuring roles to reflect an organization's lines of authority and responsibility. More powerful (senior) roles are placed toward the top and less powerful (junior) roles toward the bottom of the diagrams representing them ([19]). To limit the scope of inheritance in hierarchies, a special kind of roles is used, named private roles ([18]). Permissions are approvals of particular modes of access to objects in the system. Permissions are always positive and confer on their holder the ability to perform an action in the system ([19]). Constraints are another important aspect of RBAC and are sometimes argued to be the principal motivation for RBAC. A common example is that of mutually exclusive roles. Constraints are a powerful mechanism for enforcing higher level organizational policy. Once certain roles are declared mutually exclusive, there's less concern about assigning individual users to roles ([19]).

According to [18], RBAC provides a means for articulating policy rather than embodying a particular security policy. Hence, RBAC could be used for the redefinition of mechanisms of the original MEDAC security policy, by extending its role-based concepts and replacing stored mandatory security labels with derived ones. In this way the enhanced MEDAC (eMEDAC) security policy should utilize fully the RBAC features to support the changing administration needs of healthcare organizations.

3.2. Overview of the eMEDAC Model

The eMEDAC model uses RBAC features that have been tailored to meet the specific needs of a healthcare information system. In such a context, the access-matrix could be implemented with the RBAC permission mechanism to support the DAC features needed. In order to satisfy the MAC requirements for accessing sensitive patient data, the role hierarchy and constraint mechanisms are used in the following way.

Because there is a strong similarity between the concept of a security label (consisting of a security level and a set of categories) and a role ([18]), security levels could be implemented by means of the position (depth) of a role

(node) in the hierarchy and categories could be derived from the ancestor nodes reached when moving towards the top of the hierarchy. As a result of this, security labels are not stored but are directly derived from the hierarchy.

However, there is a need, e.g. for the medical and the ward activities of a Health Information System (HIS), to protect in a strict way the privacy of patient personal and medical data ([9]). This requirement results in a definite number of required (mandatory) security levels. Therefore, each role in the hierarchy is assigned to a specific level number that cannot exceed a predefined limit. This is a constraint that dominates the construction of hierarchies and is useful mainly in a decentralized access control system (as discussed in more detail in section 4).

In order to solve the problem of assigning to a given role a level number that is not necessarily equal to the level number of its ancestor minus one (e.g. the role is of level 2 and its ancestor is of level 4) we use dummy nodes between the two hyper nodes in the hierarchy.

Every hierarchy has at least one level of initial nodes that are assigned to the highest (or lowest) security level. Every other node is a descendant of its ancestor node and is placed in a different level. As a result of this, the category set of a node could be the set of all its ancestors (in case they are not dummy nodes) that are nearest to the top of the hierarchy (first ancestors).

Until now we assumed that each node is assigned to a different level than its ancestor node. As a result, going down a level in the hierarchy the level number is changed by one. However, this fact introduces restrictions when for example the security lattice has depth five and the role hierarchy has depth ten. To solve this problem we propose a special connection between nodes of the same level, named link.

To support the above features the concept of the Hyper Node Hierarchy (HNH) mechanism has been introduced.

3.3. The Hyper Node Hierarchy Mechanism

A Hyper Node Hierarchy (HNH) is a set of nodes and connections (figure 2). Each (regular or dummy) node is connected to another node by a branch or a link. A branch is the connection of a node to its ancestor node in the above level of the same HNH. Links are connections that are used between nodes of the same level. A link is used in order to specialize the inner structure (in the form of a sub-hierarchy) of a hyper node, by introducing its

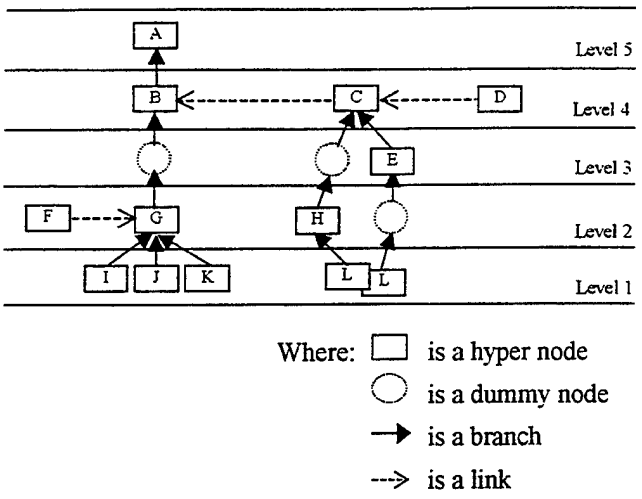


Figure 2. The Hyper Node Hierarchy (HNH).

descendant starting from the same level (otherwise branches could be used).

The HNH mechanism provides totally ordered hierarchies and supports multiple inheritance by using multiple occurrences of the same node (figure 2). A formal definition of the HNH mechanism is given below.

Definition 1: The HNH mechanism:

- N, a set of nodes
- C, a set of connections
- $HN \subseteq N \times C$, each hyper node HN is a double $\{N, C\}$,
- $HNH \subseteq HN \times HN$, is a totally ordered hyper node hierarchy.
- HN and DN, disjoint sets of (regular) hyper nodes and dummy nodes respectively,
- BC and LC, disjoint sets of branches and links respectively,
- a node N_i has a level (depth in the hierarchy) of number i,
- $BC : N_i \rightarrow N'_{i \pm 1}$, branch is a function mapping a node to its ancestor node at the above level,
- $LC : N_i \rightarrow N'_i$, link is a function mapping each node to its ancestor (hyper) node at the same level,

The security labels (consisting of a security level and a category set) are implemented in a HNH as defined below.

Definition 2: Implementation of the security level and the category set in a HNH:

- a hyper node H_i has a *security level* of number **i**,
- the *category set* of a hyper node H_i is consisted of all its possible first ancestors.

The HNH concept is used to construct user role and data set hierarchies.

3.4. The User Role Hierarchy

More specifically, the User Role Hierarchy (URH) consists of a HNH having at least one level of hyper nodes with connection "All Users". Then, these roles may be specialized into one or more levels of (senior) user roles.

The number of levels in a URH is predefined depending on the granularity of the control needed (for example in military environments it is considered to be four, respectively for Unclassified, Confidential, Secret, and Top Secret). User roles placed on the top of the URH are of lowest level (e.g. 1).

By its turn, the use of links introduces new sub-hierarchies where the same procedure can be repeated until the security administrator is satisfied that the specialization achieved is enough for the specific application. The number of new sub-hierarchies resulting from the use of links of hyper nodes is not predefined, allowing the security administrator the ability to analyze any hyper node, reaching to any depth of analysis in the tree, depending on the special needs of the application. For example in a specialized hospital (e.g. cancer hospital) the set of user roles needed is more sophisticated than in a general hospital. However, each sub-hierarchy resulting from a hyper node (to analyze further its inner structure) has a predefined number of levels that is derived from the level of the source hyper node. This sub-hierarchy cannot include levels lower than the level of the source hyper node (e.g. it includes only levels 3 to 5 for a source hyper node level of 3).

Concerning the derivation of the clearance level and the category set of a given user role, we always initialize the level to be one (lowest level) and the category to be the empty set. Then, in the hierarchy we find the first entry (occurrence) of the role and we move towards the top of the hierarchy by following its connection to its ancestor node. In case it is a branch, we increment the clearance level. In case the ancestor node is a regular node we assign it to the user role category. When the connection is 'All Users' we stop moving and we add the last category found to the category set. The same procedure is repeated for every subsequent occurrence of the user role.

The derivation of the security (clearance) level of a user role UR is described in the following algorithm 1.

Algorithm 1: Derivation of the security level.
security_level : = *lowest_level*
until connection(UR) = 'All Users' **do**
  find UR
  if type_of_connection(UR) *is branch*
    then security_level : = *security_level* + 1
  UR : = *connection*(UR)
od

The following algorithm 2 describes the derivation of the category set of a user role UR.

Algorithm 2 Derivation of the category set.
find UR
category_set : = fcs(UR)

function fcs(A)
fcs : = {}
if type_of_node(A) *is hyper then* fcs : = *node*(A)
if connection(A) ≠ 'All Users' *then*
  Anc : = *connection*(A)
  find Anc
  if fcs(Anc) ≠ {} *then* fcs : = fcs(Anc)
  findnext A
  while A *exists* **do**
    fcs : = fcs ∪ fcs(A)
    findnext A
  od
fi
fend

The use of the URH is different from the use of a classical RBAC hierarchy. In the second layer of the eMEDAC model (figure 1), the URH is used to inherit permissions. However, in the third layer the URH is used for deriving security labels consisting of clearance levels and category sets. As a result, the two basic principles of the MAC model (18]), which are the simple-property (or read-up protection) and the *-property (or write-down protection), are satisfied in a subsequent step by the secure DBMS implementing the following dominance relationship. A security label $S_1 = (L_1, C_1)$ dominates a security label $S_2 =$

(L_2, C_2) if and only if $L_1 \geq L_2$ and $C_1 \supseteq C_2$, where L is the security level and C is the category set. ([2]).

In order to demonstrate the above concepts we describe below a representative subset of our pilot implementation. The general model of a HIS architecture that has been defined by the ISHTAR group ([1]), has been used as the basis of the implementation since it corresponds to the structure of the AHEPA General Hospital that is used as our test-bed side. The four main categories of activities defined in this model have been used also as the user role and data set categories : ward, medical, administration and logistics.

In our implementation, the following user roles have been identified ([12], [13]).

User roles	Abbrev.	User roles	Abbrev.
Doctor	D	Nurse	N
Personal doctor	DP	Head Nurse	NH
On duty doctor	DO	On duty nurse	NO
Head doctor	DH	Training nurse	NT
Medical	M	Ward	W
Administration	A	Logistics	L

Table . User roles defined.

Using the specialization and generalization concepts the URH shown in figure 3 has been constructed.

This User Role Hierarchy example can be implemented using the data structure that is shown in the following table 2.

Type of Node	Node	Type of Connection	Connection
hyper	A	link	All Users
hyper	L	link	All Users
hyper	W	branch	All Users
hyper	M	branch	All Users
hyper	N	link	W
hyper	NT	link	N
hyper	NO	branch	N
hyper	NH	branch	01
dummy	01	branch	N
hyper	D	branch	02
dummy	02	branch	M
hyper	DO	branch	D
hyper	DP	branch	D
hyper	DH	branch	D

Table **2.** The data structure used for the URH example.

To derive the security label (consisting of a clearance level and a set of categories) of the user role NH we initialize the level to be one. We first find in the data structure the entry of node NH. Then we follow its connection that is a branch to the dummy node 01 and the level is incremented by one. Again, following the branch of dummy node 01 to the user role N, the level is incremented by one. Then we follow the link of N to W. Because the connection of W is 'All Users' we stop moving, the node W is included in the category set and the level is incremented by one having now the final value four.

3.5. The Data Set Hierarchy

The sensitivity of data sets is expressed in a similar Data Set Hierarchy (DSH). The same as above procedures are applied, except that data sets placed on the top of the DSH are of highest level (e.g. 5). Each sub-hierarchy that is generated for analyzing further the inner structure of a data set has a predefined number of levels and cannot include levels higher than the level of the source data set (e.g. it includes only levels 1 to 3 for a data set level of 3). Cover stories could be implemented by using the concept of private roles ([18], but for data.

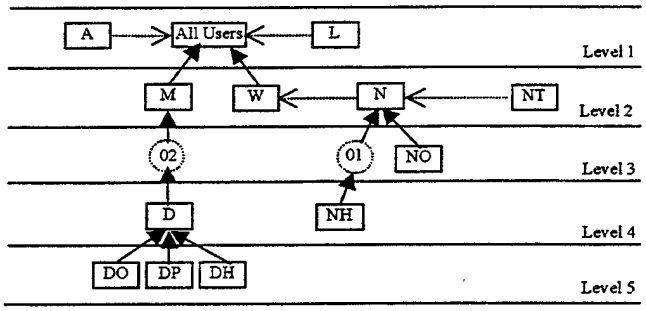


Figure **3.** The User Role Hierarchy (URH) example.

Concerning the derivation of the sensitivity level and the category set of a given data set, we always initialize the level to be five (highest level) and the category to be the empty set. Then, in the hierarchy we find the first entry (occurrence) of the data set and we move towards the top of the hierarchy by following its connection to its ancestor node. In case it is a branch, we decrement by one the sensitivity level. In case the ancestor node is a regular node we assign it to the data set category. When the connection is 'All Data' we stop moving and we add the last category

found to the category set. The same procedure is repeated for every subsequent occurrence of the data set.

The derivation of the security (sensitivity) level of a data set DS is described in the following algorithm 3.

lgorith **3** Derivation of the security level in DSH.
security_level : = *highest_level*
until connection(DS) = 'All Data' **do**
  find DS
  if *type_of_connection*(DS) *is branch*
    then *security_level* : = *security_level* - 1
  DS : = *connection*(DS)
od

The algorithm for the derivation of the category set of a data set DS is identical to the algorithm 2 (defined for user roles).

The data sets are allocated at each node depending on the need to know principle of user roles and the specific security constraints that are implemented by the secure design methodology. The following data sets have been identified for the pilot implementation ([12], [13]).

Data Sets	Abb	Data Sets	Abb
Administration	A	Physiotherapy	HTP
Patient Identification	I	Pharmaceutical	HTF
Patient Demographic Data	IN	Radio Treatment	HTR
Patient Social Data	IS	Exam Orders & Results	HE
Insurance Data	II	Clinical	HEC
Patient Hospitalization	H	Laboratory	HEL
Medical Decisions	HD	X-ray	HEX
Diagnoses	HD	Logistics	L
Diagnoses (like HIV)	HD	Cost-accounting	LC
Expectations	HDE	Administrative data	LA
Therapeutic Treatments	HT	Medical	M

Table 3. Data sets defined.

Using the above data sets the DSH shown in figure 4 has been identified for the pilot implementation.

4. Extensions for Distributed Medical Databases

Until now, the discussion addressed mainly the needs of centralized database systems. However, the presented eMEDAC security policy does not take under consideration any additional requirements for distributed database systems where centralized administration of

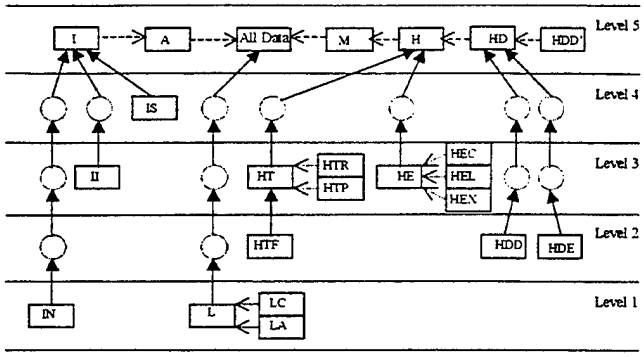


Figure 4. The Data Set Hierarchy (DSH) example.

access rights is a very difficult task. In such a case, access control systems might allow administrative authority for a subset of objects to be delegated by the global security administrator to other local security administrators. For example, authority to administer objects in a particular region could be granted to the regional security administrator. Control over the regional administrators could be also centrally administered, but they could have considerable autonomy within their regions. The construction of a local HNH (sub-hierarchies for local user roles and data sets) could be accomplished by the regional administrator without having the possibility to assign security levels higher than the corresponding global user roles and data sets that have already defined by the global security administrator. This process of delegation could be repeated within each region to set up sub-regions and so on ([15]).

However, because of the specific needs of the distributed systems design and operation, the introduction of new proposals or the extension of the already existing ones in eMEDAC is required. In our opinion, the main issue is the location of users and data sets. We have already studied the allocation of data sets in the context of a secure distributed database design methodology ([8]). However, we believe that it is worth to study the impact of different user locations in the determining process of access control for at least two things. First, knowing the location (site or administrative domain) from where the user is accessing remotely the database system, and having previously defined the trustworthiness of this location, the decision of what set of roles he can activate depends, besides his task requirements, on the trustworthiness of his current location. Second, knowing the location (administrative domain) wherein the user is acting, the decision of what kind of

access is allowed to him depends not only on his current set of roles but also on its particular user location.

In such a context, we consider the use of global and local access control mechanisms in distributed medical database systems. As has been proposed in [20], global roles could be authorized with global permissions to access an intranet's resources on the basis of different local roles in different servers (sites). However, this assumption introduces lack of flexibility (e.g. the granularity used when defining the permission set) and alike behavior of the access control system on local and remote access requests. We prefer to define separately each one subject (global or local user role) and its permission set to access remotely any defined object of the system from any defined location. Our perspective is to reduce authorized privileges of a given user role while its location is going farther on either on organizational or security level.

As with user roles and data sets, a user location hierarchy is resulting. The User Location Hierarchy (ULH) is a means for representing the organizational structure of the healthcare establishments involved in the application. Furthermore, in a ULH can be included national or international administrative domains, as well as a number or all the possible workstations from where a given user can log in the distributed medical database system.

As a result of this the total amount of specifications for defining all the needed data can easily become huge and very difficult to manage. This huge amount of data can be reduced however dramatically by exploiting the inheritance in the already presented three types of hierarchies.

In each administrative domain the following actions should be accomplished for the definition of those hierarchies:

- Inherit catalogues and hierarchies of user roles, data sets and user locations from the upper (global) administration domain,
- Refine each hierarchy to meet the special local needs of the specific administrative domain.

However, to avoid the possibility that local security administrators could decide about the authorization of subjects of their administrative domain on objects that belong to other domains, it is obvious that there must be a limit in the penetration that other administrators can do in the access control policy of each administrative domain ([3]). This can be accomplished by eliminating the user role permissions set on the basis of the assigned set of user locations. To achieve this a third dimension, concerning the user location, could be added to the classical access matrix.

In our oncoming DIMEDAC policy, which is an extension of the eMEDAC security policy for distributed database systems, we have attempted to cover in a satisfactory way those needs. However, this is beyond the scope of this paper.

5. Conclusion and Future Work

The eMEDAC security policy uses RBAC components to provide an enhanced redefinition of a number of mechanisms of the already known MEDAC security policy. The main innovation of this paper when compared with our previous MEDAC papers is related to the use of RBAC mechanisms in all the three layers of the original MEDAC security policy.

As explained in the paper, the use of RBAC mechanisms in the third layer raises several problems that we addressed. For example, the construction of totally ordered security levels and the definition of different number of security and hierarchy (depth) levels. As a basic solution to the problems raised we proposed the Hyper Node Hierarchies (URH and DSH hierarchies of any depth, derived security levels and categories, access control data stored separately from the application data holders, etc.) as a unique mechanism for inheriting permissions (discretional control) and deriving security labels (mandatory control). The proposed concept of the HNH mechanism provides a flexible number of refinement levels that is not strictly equal to the number of mandatory security levels.

Because MAC, DAC and RBAC are supported in eMEDAC at the same time, access control becomes more efficient and hence particularly suited to security critical environments, like the health care ones, where all three aspects of security are important.

Administration also becomes easier because the eMEDAC policy, besides the utilization of RBAC administration advantages, provides the ability of mixed centralized and decentralized administration control. The Data Set Hierarchies are centrally defined and stored separately from the data holders. In a distributed system both URH and DSH can be centrally defined and replicated to each site. The local administrators can add more refinement levels for satisfying their local needs. However, the HNH concept specifies (in a mandatory way) a certain number of security levels that cannot be overridden.

A possible disadvantage is a non-significant, to our opinion, loss in performance (or higher computational workload) which stems from the need for deriving instead of using already stored security labels. However, there is expected to be a significant saving of storage space because, instead of storing security labels for every object (depending on the selected granularity, e.g. for field granularity the storage cost becomes considerable high), only the necessary nodes of HNH are stored.

In our future work we are going to investigate and compare the computational and storage cost of performing access control by deriving security labels instead of using stored ones. We also are working to study the extensions of the eMEDAC that are necessary in distributed database systems. Furthermore, the case of distributed systems with mobile parts is expected to introduce a number of new factors affecting the overall access control system.

6. References

[1] Blobel B., Bleumer G., Muller A., Flikkenschild E. and Ottes F., 'Current Security Issues Faced by Health Care Establishments', *in HC1028 Implementing Secure Healthcare Telematics Applications in Europe (ISHTAR)*, October 1996.

[2] Castano S., Fugini M., Martella G., Samarati P., 'Database security', *Addison Wesley publishing company*, 1994.

[3] Ceri S. and Pelagatti G., 'Distributed Databases: Principles and Systems', *McGraw-Hill, NY*, 1985.

[4] Essmayr W., Kapsammer. E., Wagner R.R., Tjoa A.M., 'Using Role Templates for Handling Recurring Role Structures', *Proceedings of the 12th International IFIP WG11.3 Working Conference on Database Security*, Chalkidiki, Greece, July 1998.

[5] Ferraiolo D. and Kuhn R., 'Role-Based Access Control', *in Proceedings of 15th National Computer Security Conference*, 1992.

[6] Ferraiolo D., Cugini J. and Kuhn R., 'Role-Based Access Control (RBAC): Features and Motivations', *In Annual Computer Security Applications Conference, IEEE Computer Society Press*, 1995.

[7] Khair M., 'Design and Implementation of Secure Database Systems with Application on Heatlhcare Information Systems', *PHD Dissertation*, 1996.

[8] Khair M., Mavridis I. and Pangalos G., 'Design of secure distributed medical database system', *in Database and EXpert systems Applications, DEXA '98*, August 1998.

[9] Pangalos G., 'Medical Database Systems Security', *in EEC/AIM, SEISMED (A2033) Project Report, No. AIM/SEISMED/SP-07/20-04-95/3*, 1995.

[10] Pangalos G., 'Secure medical databases', *Proceedings IMIA security conference, Finland*, 1996.

[11] Pangalos G., Khair M., *'Design of a secure medical database systems'*, in IFIP/SEC'96, 12th international information security conference, 1996.

[12] Pangalos, G., Gritzalis D.; Khair, M. and Bozios, L., 'Improving the Security of Medical Database Systems', *in Information Security - the Next Decade, J. Eloff and S. Von Solms editors, Chapman & Hall*, 1995.

[13] Pangalos, G., Khair, M. and Bozios, L., 'An Integrated Secure Design of a Medical Database System', *MEDINFO'95, The 8th World Congress on Medical Informatics, Vancouver, Canada*, 1995.

[14] Poole J., Barkley J., Brady K., Cincotta A. and Salamon W., 'NISTIR 5820 Distributed Communication Methods and Role-Based Access Control for Use in Health Care Applications', *can be found in* http://www.itl.nist.gov/div897/staff/poole/documents/nistir5820.htm

[15] Sandhu R. and Samarati P. 'Authentication, Access Control, and Intrusion Detection', *The Computer Science and Engineering Handbook*, 1997.

[16] Sandhu R., 'Access Control: The Neglected Frontier', *In Proceedings of First Australian Conference on Information Security and Privacy, Wollongong, Australia*, June 23-26, 1996.

[17] Sandhu R., 'Rationale for the RBAC96 family of access control models', *in Proceedings of the 1st ACM Workshop on RBAC, MD, USA*, 1996.

[18] Sandhu R., 'Role-Based Access Control', *Advances in Computers, Vol.46, Academic Press*, 1998.

[19] Sandhu R., Coyne E., Feinstein H. and Youman C. 'Role-based access control models', *IEEE Computer*, 29(2):38-47, February 1996.

[20] Tari Z., and Chan S.-W., *A Role-Based Access Control for Intranet Security*, IEEE Internet Computing, September – October, pp.24 – 34, 1997.

[21] Vandenwauver M., Govaerts R. and Vandewalle J., 'Role Based Access Control in Distributed Systems', *Communications and Multimedia Security Vol.3, pp.169-177*, 1997.

Agent Approaches to Enforce Role-Based Security in Distributed and Web-Based Computing*

S. A. Demurjian, Sr., Y. He, T. C. Ting, and M. Saba

Department of Computer Science and Engineering
The University of Connecticut
191 Auditorium Road, U-155
Storrs, CT 06269, USA
Email: {steve,ting,saba}@engr.uconn.edu
Tel. 860-486-3719

Abstract

In the age of information technology, organizations of all types are seeking to effectively utilize and disseminate information, by designing and developing dependable and secure distributed computing environments that allow existing and future systems to inter-operate. While many existing access control approaches (mandatory, discretionary, and role-based) can be leveraged for the support of security in distributed and web-based settings, their assumptions of a centralized computing model may be insufficient in a distributed setting. In recent years, agent computing has emerged as a new computing paradigm, particularly suited to distributed and web-based applications. This paper explores software agents, focusing on their ability to support role-based security in a dynamic, object-based setting which is suitable for distributed and web-based applications. The agent approaches differ in their utilization of agents (stationary and mobile) and the granularity level of the involved classes/objects. We also report on our prototyping efforts using aglets, a Java-based mobile agent model from IBM.

1. Introduction

Today's and tomorrow's distributed and web-based applications will be comprised of existing legacy, commercial-off-the-shelf (COTS), and database applications that interact with new clients, servers, and web-information repositories, as organizations strive to allow information to be utilized in new and innovative ways. The fundamental challenge facing organizations, software architects, system designers, and application builders (i.e., stakeholders) involves the ability to leverage computing and networking resources, and data and information in existing applications, to construct new dependable and secure distributed and web-based applications. Our focus is on the secure nature of distributed and web-based applications, particularly in venues which promote electronic banking, commerce, information dissemination (push and pull), and so on. Distributed and web-based applications will require researchers and practitioners to design security solutions that expand and transcend available alternatives. Traditional alternatives such as *mandatory access control (MAC)* [Keef88, Land84], *discretionary access control (DAC)* [Loch88, Sand96], and *role-based security* (RBS) [Demu97] may all be useful to some degree as stakeholders and security engineers work towards the establishment of a cohesive security policy. Agent computing, which first emerged just five years ago [Gene94], has great potential for supporting the design/development of secure distributed and web-based applications.

Agents act on the behalf of individuals (users), in individual or collaborative fashion, to assist in a particular task, hopefully making it easier for users to accomplish what they intended. Software agents, as computing objects have a specific function or responsibility to perform, and can be defined in

* The work in this paper has been partially supported by a contract from the Mitre Corp. (Eatontown, NJ) and a research grant by AFOSR.

formal terms to have a state and a behavior within the runtime environment. Software agents have four mandatory properties [Lang98]: ability to sense and react to changes in the environment; autonomous control over its own state and behavior; proactive to achieve specific goals (typically of the user); and, constantly executing within the runtime environment. Stationary agents are restricted to a single computing node in accomplishing their tasks. Mobile agents can migrate to a new location in order to execute their required responsibilities. All agents are like other objects in that they can be created and destroyed. However, agents cannot interact by invoking each others methods; rather, they communicate via message passing.

The autonomous and mobile nature of agents make them attractive for security purposes. For example, one scenario could have agents dynamically created from a client application to carry out the secure access to objects across a network. In such a scenario, the agent may go forth and visit multiple nodes to collect/update relevant objects. Alternatively, multiple mobile agents may be simultaneously dispatched, each processing a single request, to achieve parallel processing in a distributed setting. Security can be included as part of the agent functionality, providing a specificity of the role of the client that dispatched the agent. Note also that mobile agents are a significant security concern from an execution perspective, due to their potential ability to act as a threat, and that there must also be protection of the agent's state from a malicious host. These two critical issues are beyond the scope of this paper.

Our intent in this paper is to present and explore various scenarios for agent approaches to security within a role-based context for distributed and web-based applications. We view this as a crucial and important first step in understanding the different ways that agent computing models can be utilized to support role-based security. Moreover, if agent computing continues to increase in popularity and usage, it is incumbent upon the research community to provide security solutions. While the technology is dangerous from a security perspective, it is critical that we offer strong and proven solutions to guarantee varying levels of secure access in a distributed setting that utilizes agents. The research community must react to rapidly changing and newly emerging technologies.

The work presented in this paper is a progression of our own efforts [Demu97, Demu98, Smar99], with the justification to pursue this effort influenced by other researchers [Hale96, Hale98, Tari98]. In our previous work [Demu98], we explored software architectures alternatives that utilized a client/server paradigm to explore role-based security. While there was distribution in some of the alternatives, the emphasis was on relatively static architectures, which is inadequate for true dynamic, distributed and web-based applications. There have been efforts to investigate security for distributed objects [Hale96], which has recently been extended to provide a secure distributed object and language programming framework that is suitable for internet-based applications [Hale98]. Another effort has utilized the Distributed Object Kernel (DOK) project as the underlying framework upon which software agents can operate for the design and implementation of distributed security policies [Tari98]. Our most recent effort has explored the capabilities and the potential impact of Java on security for distributed computing [Smar99], which we used as a starting point to explore agent approaches to role-based security, which is the main emphasis of this paper.

The goal of this paper is to explore the potential of agent computing in support of role-based security in a dynamic, object-based setting. For security in distributed and web-based applications to succeed, it is important to leverage existing security techniques in conjunction with emerging approaches to arrive at a solution for the 21st century. The remainder of this paper is organized into three sections. In Section 2, we detail and compare three architectures of agent approaches for role-based security in a dynamic, object setting. In Section 3, we explore our experimental prototyping efforts with one of the architectures presented in Section 2 to clearly demonstrate the feasibility of agent-solutions. In Section 4, we conclude by offering insight into ongoing efforts and future plans. Finally, note that issues related to the definition and management of a security policy, which is defined for both clients and servers in a distributed setting, in not addressed herein; rather, it is the subject of future work.

2. Agent Approaches to Role-Based Security

The purpose of this section is to present a series of agent approaches that can be utilized to facilitate the

role-based access of remote objects by users in distributed and web-based applications. Security for such applications must occur in an environment that may consist of legacy, COTS, database, and new/existing server applications, with clients interested in obtaining access to the remote objects that reside in these various applications. One interesting aspect of security is that its goals are often orthogonal at some level to the goals of distributed computing and web-based computing. Security has the goal of controlling and limiting certain types of interactions while distributed and web-based computing is concerned with enabling interoperation and facilitating access to information. Stakeholders must carefully balance the sometimes contradictory goals of security and distribution requirements when constructing distributed computing applications. Many organizations are actively distributing their computing and offering new ways to access both old and new information, without regard to potential security breaches or the inadvertent dissemination of sensitive information.

For discussion purposes within this paper, we are assuming a client/server computing model for distributed and web-based applications. Specifically, in our client/server model, a client application is interested in sending a request to access/modify a remote object, with the allowable actions on the remote object dictated by the role of the user (client). In our role-based model for DAC [Demu97], a customizable public interface is promoted, that appears differently at different times for specific users. We have proposed the concept of a potential public interface for all classes, which contains those methods that are most likely to be made public on a role-by-role basis. The methods that each role needs from each application class defines the security requirements for that role. The basic premise is that, for each role, we track the positive permissions, namely, can a user role invoke a particular method on an object. Collectively, all of these security requirements form a foundational part of the security policy for an application. Our assumption in this paper is that the client will play a role and make a request to invoke a method on a remote object. Agents will be dispatched to attempt to carry out that request, with a success or failure result. The interested reader is referred to our efforts at UConn on extensible and reusable DAC enforcement mechanisms for a detailed overview of our role-based security model [Demu97]. This expertise has been crucial in formulating the architecture agent variants presented herein and in prototyping the agent-based approaches presented in Section 3.

Thus, the scenario is that a client application is making a request (i.e., a method invocation) to access/modify a remote object, that may be residing in a legacy, COTS, database, or server application. In such a scenario, we want the request to be processed according to the defined security policy for the client application in a dynamic setting. In our role-based security model, this requires that, for each user role, we maintain a list of the permissions (which methods can be invoked) on all classes in an application. Agents are attractive in such a setting, since they allow autonomous and mobile actions to occur on behalf of a client. In our case, these actions will authenticate the identity and role of the client application, will bundle the client request (method invocation) and user role within a message, and then dispatch agents that can move across computing nodes for the secure access of remote objects.

In this section, we present three agent approaches that support role-based access for distributed and web-based applications:

- A *baseline agent approach* that represents a core set of system components and agents (both stationary and mobile) to facilitate the remote access to an object.
- A *hierarchical agent approach* that expands on the baseline approach to spawn a series of multiple agents for simultaneously accessing remote objects in multiple locations.
- An *object-security manager agent approach* that extends the hierarchical approach to include the access of multiple remote objects in the same location.

The remainder of this section reviews each approach in detail, followed by a summary that compares the three approaches. Finally, note that while we have employed a role-based paradigm for security, there is nothing in the ideas presented in this section that will restrict our solution.

2.1 Baseline Agent Approach

The architecture for the baseline agent approach is presented in Figure 1, and is partitioned into a client computing node (upper half of figure) and a server computing node (lower half of figure). The components and agents are as follows:

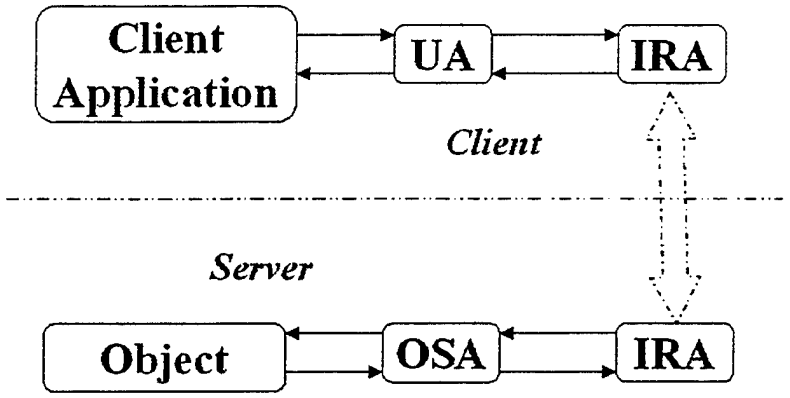


Figure 1: Architecture for Baseline Agent Approach.

- **Client Application (CA)**: This component is the graphical interface or software tool that is utilized by the user to perform a specific set of tasks. The user is restricted to playing a single role at any time, and both the role and request are passed from CA to the user agent (UA) for processing. Users can access/modify only a single remote object at a time via CA. If multiple remote objects are to be accessed/modified, there must be functionality within CA that allows such a request to be separated into a series of individual object accesses. Agent approaches in Sections 2.2 and 2.3 will consider requests that involve access to multiple remote objects.
- **User Agent (UA)**: This is a stationary agent that is created upon the request of the client to represent the user. UA receives a request from the client, transforms the request to the proper format, creates the information retrieval agent (IRA), forwards the request to IRA, waits for the processing of the request, collects the results of the request from IRA, and transforms and then returns the results to CA.
- **Information Retrieval Agent (IRA)**: This is a mobile agent that is created by UA for processing the request of CA. In this baseline approach, the IRA is limited to interacting with the UA on the client side and object security agent (OSA) on the server side. IRA is created and dispatched by UA to access a single remote object on the server side, if permitted by the role of the user.
- **Object Security Agent (OSA)**: This can be a stationary agent (collection of security objects) or a mobile agent. In either case, OSA enforces the security policy of the object that is based on the permissible actions of each role. Each OSA is effectively providing control to a remote object. The *remote object* may be a single object or a collection of objects (aggregation). In the latter case, this is essentially a database, and OSA is serving as a layer or wrapper to interact with the collection of objects.
- **Object**: The remote object that is available to provide services to CAs. A remote object may be a single instance or a collection of related instances (e.g., a person database is a single remote object).

While the above represents an overview of the capabilities of each component in Figure 1, it is important that we examine UA, IRA, and OSA in greater detail, to fully understand the subtleties

involved in their interactions and lifetimes. In some situations, there are alternative scenarios that are plausible based on the assumptions and operating environment. These alternatives will be mentioned and critiqued throughout our discussion on the agent approaches.

The baseline agent approach represents a core strategy with significant and limiting assumptions:

1. The client formulates simple requests that involve the access of a remote object by a single method call. Each request may invoke a single method on a single remote object.
2. UA is able to process multiple requests by the client in a synchronous fashion. Thus, there may be multiple active IRAs processing requests simultaneously.
3. Authentication of the user must occur prior to the spawning of an IRA to respond to a request. We are assuming that this will take place in either CA or by UA.
4. While a *remote object* refers to a single instance, that instance may be a person record (e.g., name, address, etc.) or a collection instance (e.g., the collection of all person records). In the latter case, the single remote object contains multiple instances that are controlled and managed like a database within the collection.

The idea for the baseline agent approach is to limit functionality by mandating that complex requests that require synchronization (results of one request drive submittal of successive requests) are handled by the client. While it may be argued that this is unrealistically limiting, our intent is to provide a simple model for those situations where the majority of the processing already exists in CA, which may occur when the client is a legacy/COTS application.

2.1.1 User Agent

The *user agent (UA)* is a stationary agent that is utilized by the client application when the user requests that an action be initiated on his/her behalf. The user, via CA, can only play a single role at any given time. UA is the interaction medium between CA and the mobile IRA. The UA receives a coded request (i.e., user role and method to invoke) from CA, forwards the request to IRA, and waits for the response from IRA. For the baseline approach of Figure 1, UA receives a single request to access one remote object. There are two different UA allocations strategies:

- *User-Based Allocation*: In this situation, a UA is allocated and dedicated to an individual user when the client application commences and the user logs on to a session. Essentially, a dedicated UA can enforce the single role that the client plays and manage all of the synchronous requests that will be submitted to dedicated IRAs (one IRA per request). If multiple CAs are executing on the same client node, then multiple UAs will compete for computing resources.
- *Role-Based Allocation*: In a multi-user environment, where multiple users are active, it makes sense to explore an allocation strategy that is based on role. In this situation, dedicated UAs are allocated for each role, and shared by the many users (CAs) that are playing the same role. UAs are allocated whenever there is a request for a user (CA) to play a role, and that role is not supported by an active UA. When a UA is no longer being utilized by any active CA, then it can be de-allocated.

In both UA allocation strategies, multiple IRAs may be active and managed (i.e., created, dispatched, and destroyed) by the single UA; the difference is that in user-based allocation there is one client/UA, while in role-based allocation there is one UA/role. The allocation above is based on the assumption that a user (CA) can only play one role at any given time. If a user can play multiple roles simultaneously, user-based allocation is still feasible, since each UA will embody the security policy for the multiple roles being played. However, role-based allocation becomes problematic. Since the security privileges for a user are spread across multiple UAs, the UAs would be required to communicate to synchronize their activities. Thus, the lifetime of UA can vary based upon different scenarios of CA behavior within a shared client computing node.

2.1.2 Information Retrieval Agent

The *information retrieval agent (IRA)* is a mobile agent that is created by UA on the client side to process the request of the user (from CA). The purpose of the IRA is two fold: (1) separation of concerns to allow the IRA to be mobile and interact with remote objects and (2) to exploit parallel and distributed processing by having multiple IRAs active to process requests (invoke methods) for a given CA

and managed by a UA. In the baseline agent approach, the request to be processed by IRA will be limited to a single remote object. As a mobile agent, IRA is able to move to the host computer (server side) where the object resides that is needed to satisfy the user's request. IRA carries the user request (which method to invoke) and the current role, which are both needed to interact with the object-security agent (OSA) to process the request. That is, the OSA will validate whether the IRA is permitted to ask for the method to be invoked on the remote object based on the user role being represented by the IRA. In the baseline agent approach, IRA moves to the destination and exchanges information with OSA. Once the IRA has received a response (success or denied access) from OSA, it takes that response, ceases its action, and moves back to the client side to return the result to UA, which will then forward the response to CA. As in Section 2.1.1, there are two different scenarios for the lifetime of IRA that share a common basis, IRA is allocated by UA for the first request of a remote object by CA.

1. IRA stays alive as long as UA is active. In this case, a single IRA can process all requests by a client in a sequential manner, starting a new request only after the current request has been completed. This scenario reduces the overhead associated with creating and destroying a mobile agent.
2. IRA is de-allocated when it finishes processing a request. In situations were access to remote objects by clients is infrequent, this scenario benefits by not having active, idle agents.

The first scenario will not support multiple active requests being spawned by UA for simultaneous processing by IRA. The second scenario supports such a situation, since multiple independent IRAs can be spawned that move to remote locations to access objects. This will require UA to have additional logic to be able to process a queue of simultaneous requests from the same or different users.

Finally, note that in the baseline agent approach and the others to be discussed in Section 2.2 and 2.3, we are making the assumption that an IRA processes one request to invoke one method on a single remote object. The intent of this assumption is to allow the IRA to be lean or thin, targeted to a very simple and easy to carry out task. This assumption is in line with the idea that the baseline agent approach only processes simple requests. There may be multiple IRAs active at the same time for a single UA, working on individual requests that will each invoke a single method on a single remote object. Having a single IRA travel to multiple locations isn't appropriate for the baseline agent approach, but may be relevant for the other approaches, as we will discuss in Sections 2.2 and 2.3.

2.1.3 Object Security Agent

The *object security agent (OSA)* can be conceptually regarded as the firewall that separates the remote object from the outside world. OSA embodies the security policy for the remote object, which in our case, is a role-based approach that dictates which role can access which method on the object [Demu97, Demu98]. OSA receives a request to access a remote object via the mobile IRA, and based on the privileges of the role, will either deny or respond to the request. For our purposes, OSA must validate the privileges carried by IRA (user role and method to invoke) against the applications security policy for that object/class. OSA is a gatekeeper to the remote object, but is important to again stress that the remote object may not be a single instance, but may be a collection of instances. A person collection is a single instance of the collection class that contains multiple person instances. OSA may manage a single instance (the "Steve" instance) or an entire collection (the "PersonDB" instance which contains instances for "Steve", "T.C", "Ying", "Mitch", etc.). Thus, a remote object of an object oriented application is equivalent to a database.

The OSA component can be designed and implemented as either a set of security objects or a mobile agent. If OSA is a set of security objects, it is tightly bound to the class (type) of the remote object, and the supported security policy will be one of the object-oriented role-based techniques we have described in our earlier work [Demu97]. Allocation of OSA in this situation will occur when the remote object is instantiated. If the OSA is an agent, it must be a separate entity from the remote object. The security policy that is supported in this situation can occur at the class (type) or object (instance) levels. In this case, there are a number of strategies for OSA allocation:

1. When the number of remote objects on a given computing node is "small" (or the number of objects that are typically accessed is small), then

allocate a single OSA that is shared by all of the remote objects on the same server.

2. When the number of remote objects is "moderate" (or the number of objects that are typically accessed is moderate), then allocate an OSA for every remote object (instance).
3. When the number of remote objects is "large", but the number of involved classes (types) is small, allocate an OSA for each class (type) of remote objects.

In either of these strategies, OSA will contain an aggregation of the security policies, i.e., methods that can be invoked by all roles for all classes/objects. There are clear tradeoffs to be made in the aforementioned three strategies, particularly when one considers the frequency of access of remote objects by IRAs. For example, a single, shared OSA per server will quickly be overload by multiple IRA requests. Moreover, as activity increases and the numbers of mobile IRAs increase, regardless of the scenario, it is likely that another component would be needed to oversee the interactions of IRAs with multiple OSAs. This will be discussed in Section 2.3.

2.2 Hierarchical Agent Approach

The baseline agent approach was limited to simple requests that can be submitted synchronously without coordination of results. The assumption was that CA, likely a legacy or COTS application, would handle the processing of complex, nested requests. In practice, requests by clients are often complex, requiring multiple methods to be invoked in a particular sequence, similar to the concept of a nested transaction. The hierarchical agent approach is appropriate when there are thin clients (e.g., a browser) and the processing of complex requests must be offloaded to either UA or IRA.

Figure 2 contains a representation of the architectural components for the hierarchical agent based approach which was designed to handle complex requests. While similar to Figure 1, the major difference is a hierarchical collection of mobile IRAs that are able to handle complex requests submitted by the client via UA. A minor difference in the figure has the Security Policy listed as independent from OSA; in fact, as discussed in Section 2.2.3, it can be incorporated within OSA. In the baseline agent approach, the strong assumption was made that a user submits a request to access an individual remote object. Any complex, ordered request must be submitted as a series of sequential single requests, with UA handling the submittal of requests and collection of results. In the hierarchical agent approach, the logic in the IRA has been enhanced to support complex requests involving multiple remote objects on possibly different computing nodes.

To support complex requests, there are three different types of IRAs: *root-IRA*, *internal-IRA*, and *leaf-IRA*. A *root*-IRA interacts with UA to handle and oversee complex requests that involve multiple remote objects. The root-IRA is spawned by UA, and in turn can spawn other IRAs to handle complex requests. If the complex request is a series of requests that involve access to individual remote objects, the root-IRA will spawn a series of leaf-IRAs. A *leaf-IRA* is similar in concept to the mobile IRA discussed in Section 2.1.2, and is intended to process the request for access to a remote object by moving across the network to another node. A leaf-IRA interacts with OSA to answer the user request, and returns its result to the root-IRA. Figure 2 depicts a situation where there is one root-IRA and two leaf-IRAs. Each leaf-IRA takes the request to access a single remote object and moves to the appropriate computing node to interact with an OSA according to the scenario described in Section 2.1. The two leaf-IRAs in Figure 2 could move to the same or different computing nodes to accomplish their respective tasks. Upon their return, the root-IRA collects the results for forwarding to UA and eventually the user. Note that in the case where a simple request for a single remote object is made, the root-IRA may be designed in such a way as to process the request, or the root-IRA may spawn a single leaf-IRA. In the former, root-IRA would be a mobile agent, in the latter, it would be stationary.

While Figure 2 only illustrates two levels of IRAs, in fact, there can be multiple levels, based on the complexity of the user request. If the root-IRA decomposes the user request into multiple complex sub-requests, each of these sub-requests would be dispatched to an internal-IRA for processing. That is, an *internal-IRA* is spawned to handle complex sub-requests. An internal-IRA will in turn spawn either internal-IRA or leaf-IRA, to process, respectively, complex sub-requests (multiple remote objects) or simple requests (single remote object). Recursive spawning of IRAs will continue until the state of all leaf-IRAs is reached. As internal-IRAs and leaf-IRAs

complete their actions, their responses will be collected by internal-IRAs, and eventually the root-IRA. In the aforementioned situation, the root-IRA is stationary; internal-IRAs can be either stationary or mobile.

From an allocation perspective, one approach would have a root-IRA allocated for each UA, whose lifetime is under the control of UA. Internal and leaf IRAs would be created by the root-IRA as needed to process user requests. Since only one root-IRA exists for each UA, the requests from the clients can be handled in a synchronized fashion. However, unlike the baseline agent approach, CA can submit complex requests to UA which are then passed to root-IRA for processing. The internal and leaf IRAs that are spawned can process the sub-requests in a parallel fashion, by dispatching mobile agents (leaf-IRAs) to remote object locations.

Again, like the baseline agent approach, we have adopted the strategy that each leaf-IRA will process a single user request (method invocation) on a single remote object. Clearly, we could have modeled a different strategy where the single IRA can visit multiple locations to satisfy a complex request of CA. However, our top-down strategy of root-IRAs spawning internal-IRAs spawning leaf-IRAs, is more suited to having each leaf-IRA visit a single remote object, and then require the internal-IRAs to collect and synthesize results in a bottom-up manner, eventually arriving at the root-IRA. A different strategy could have had each internal-IRA spawn a single leaf-IRA that would have a list of multiple user requests requiring access to potentially multiple remote objects. The single leaf-IRA could visit multiple servers in pursuit of all relevant remote objects. The tradeoffs are as follows:

1. The single leaf-IRA for an internal IRA may slow down processing, since the internal-IRA may need to get interim results from the single leaf-IRA before the same leaf-IRA is dispatched to other nodes.
2. The multiple leaf-IRAs may slow down processing at the client side with many agents that must be tracked and managed.

There should be no impact on the server side in either situation, since the number of visits is the same; who is visiting is what has changed.

2.3 OSA Manager Agent Approach

The final architecture, the *object-security manager agent approach*, is presented in Figure 3, and expands the hierarchical agent approach by the addition of an OSA Manager. The *OSA Manager* is intended to assist in OSA allocation strategies as discussed in Section 2.1.3. OSA Manager is responsible for arranging and overseeing the allocation of the OSA. Recall from Section 2.1.3 that we can allocate OSA in a number of ways: one per server when there are "few" remote objects, one per object for a "moderate" number of objects, or one per class, for a "few" classes that have many instances. Rather than having one of these alternatives chosen in a static fashion at start up, an OSA Manager can dynamically pick and choose the best allocation strategy for controlling access to remote objects during execution, in reaction to the state of the system, i.e., number of IRAs at client, number of remote objects being accessed, etc. In some situations, an OSA Manager might have an OSA for multiple objects coexisting with other OSAs that are dedicated to specific classes. The OSA agent approach extends the hierarchical approach by adding another layer of management at the server side to insure that the collections of remote objects are being managed in a fashion that is most suited to the execution environment of the running application.

Processing by mobile IRAs must now be altered to communicate with OSA Manager to obtain information on the OSAs to be contacted for remote object access. That is, upon arriving at a server, an IRA asks OSA Manager where to find a specific remote object. OSA Manager returns the identity of the OSA that is controlling the remote object. Once the identity has been received, IRA contacts the correct OSA and proceeds to process the user request.

The biggest advantage to an OSA Manager is to have tight and dynamic control over the OSAs that are resident on a server for processing user requests. Security policies, which reflect the application environment and conditions, are ever-changing, and the OSA Manager approach allows us to easily alter the allocation strategy for OSAs. In fact, OSA Manager can maintain historical information on OSA access by IRAs, which can be utilized to terminate infrequently used OSAs, or change from instance-based to class-based allocation. In the object-security manager agent approach, OSAs don't have to be statically created at system initialization.

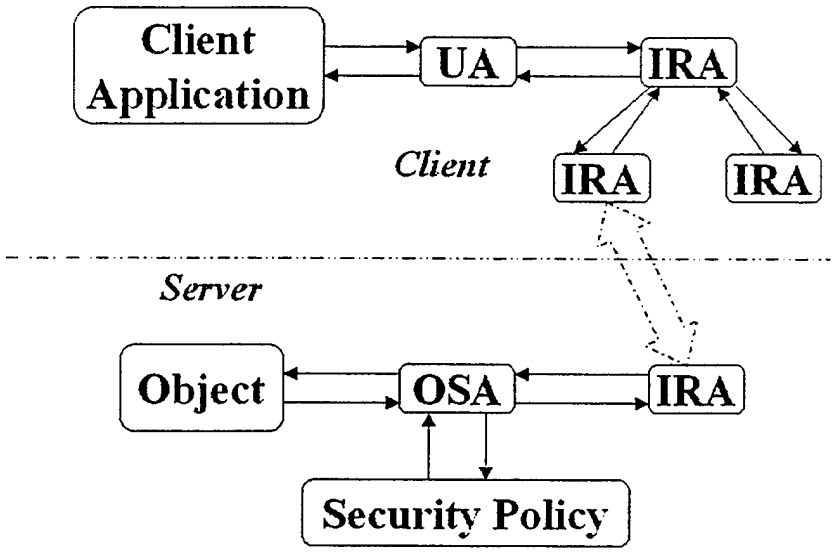


Figure 2: Architecture for Hierarchical IRA Approach.

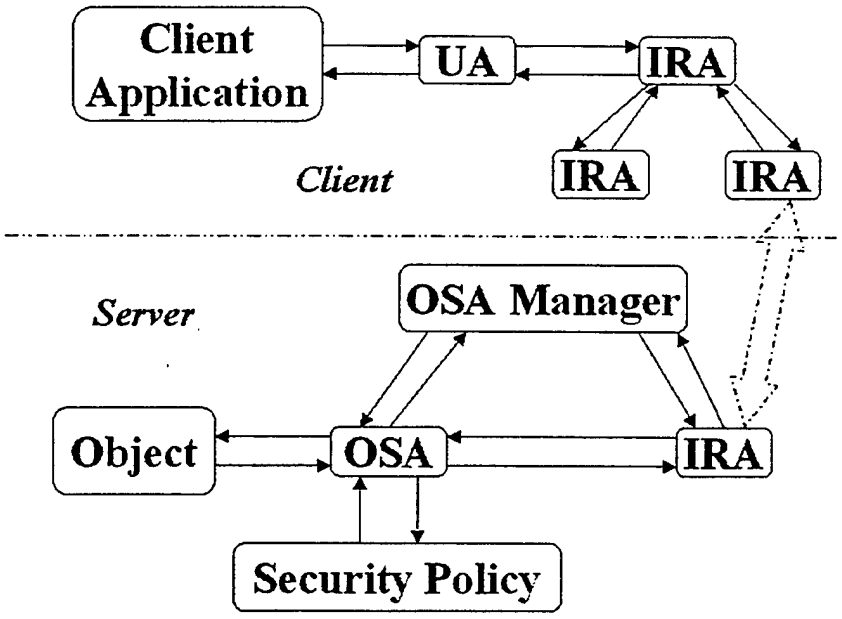


Figure 3: Architecture for OSA Manager Approach.

2.4 Summarizing Approaches: Baseline vs. Hierarchical vs. OSA Manager

The three approaches presented in Sections 2.1, 2.2, and 2.3 represent a progression based on a number of different dimensions. From the dimension of *client functionality*, the baseline agent approach is particularly well suited for legacy/COTS clients, where individual requests (method invocations) can be captured and sent to UA for processing. Since the functionality within the client in this situation is typically inaccessible (no source code), it makes sense to decompose the problem and send simple requests to UA which in turn create and dispatch IRAs as needed. The hierarchical and OSA Manager will work in this situation, but are better suited for thin clients (e.g., browsers), where significant application functionality is then offloaded to UA and IRA. From the dimension of *UA allocation*, all three approaches are able to use either user-based or role-based allocation (see Section 2.1.1 again). The different allocation strategies are based more on client activity than on the agent behavior, as was discussed. From the dimension of *adaptability of server to IRA frequency and load*, the OSA Manager approach is superior. OSA Manager is intended to adapt the allocation of OSAs to remote objects based on the access patterns of the agents, ranging from one OSA/remote object to one OSA/class to one OSA/server. In the baseline and hierarchical approach, allocation of the OSAs would be fixed at system initialization. Finally, from the dimension of *complex request processing*, the hierarchical and OSA Manager approaches are equivalent in their ability to support nested transactions. All three approaches are controlling access to remote objects, which may range from small, individual instances (e.g., the Person "Steve") to collection instances (e.g., the PersonDB collection of Persons).

3. A Java Aglet Implementation

Stationary and mobile software agents in Java are supported by a number of different agent based systems, including Aglets [Agle], Odyssey [Odys], Concordia [Conc], and Voyager [Voya]. For the purposes of our implementation efforts, we have chosen IBM's aglets, which was named by combining the terms of agent and applet. Unlike a Java applet, an aglet continues execution where it left off (upon reaching a new location). This is possible because an aglet is an executable object (containing both code and state data) that is able to move from computing node to computing node across a network. Like applets, aglet actions should be restricted to a Java sandbox. The sandbox model supports the running of untrusted code in a trusted environment so that if a hostile aglet is received, that aglet cannot damage the local machine. For applets, this security is enforced through three major components: the class loader, the bytecode verifier, and the security manager. Aglets would require the same level of security as applets. The aglets would need to ask permission from the security manager before performing operations, thus allowing the security manager to know the identity of the aglet. For the purposes of our paper, we are exploring the utilization of aglets in support of the baseline agent approach as presented in Section 2.1 and shown in Figure 1. We have prototyped two variants of the baseline agent approach, which differ in the OSA allocation strategies (see Section 2.1.3 again) and the number of involved computing nodes. The remainder of this section reviews the baseline agent approach which we have prototyped, and presents our assumptions, messaging activities, and subsequent usage of the prototype in a graduate course project.

3.1 Agent Implementation Approach

The agent implementation effort of the baseline agent approach is shown in Figure 4. The main difference is a Translator that is utilized to facilitate message passing communication between aglets. The Translator encodes the outgoing data from CA into a message (user request) that can be passed to UA, and retrieves data from the incoming message received by UA for the response to the request. In our prototype, a Translator is assigned to each CA/OSA and it is responsible for translating data for one user/remote object. In the implementation, the user's identity is included in every message. The Translator on the client side acquires the responsibility for authorization that was formerly performed by UA. The Translator on the server side includes the ability to invoke methods on remote object methods. Whenever OSA receives a request from IRA, it passes the message to its Translator. The Translator will analyze the message, check the permissions to determine if the role can access the given method, and if so, invoke the method and create a reply message for OSA that is then passed to IRA. IRA moves back to the client, communicates via a message to UA, which in turn routes the response back to CA.

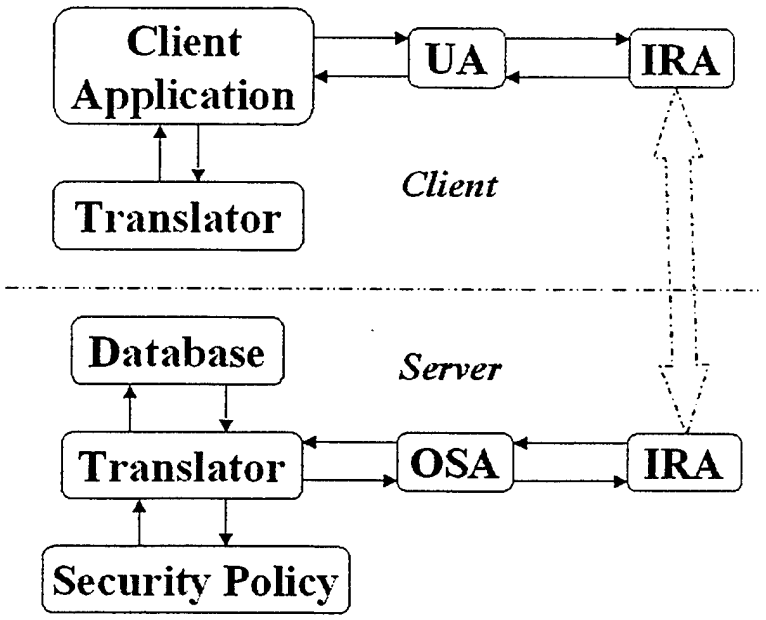


Figure 4: Architecture for Agent Implementation Approach.

3.2 Experimental Prototype

Based on the framework described in previous section, an experimental prototype was constructed. The application involves typical university database access to courses, allowing individuals to view, add, delete, register or drop courses. There are two roles supported: Faculty and Student. Faculty has the permissions to add, delete, and find courses. Student can register, drop, and find courses. There are two remote objects in the system: the Course database and the Person database. Each remote object contains data and a set of methods to operate on the data. We experimented with two variants of the architecture discussed in Section 3.1 and shown in Figure 4. The first variant placed the two remote objects, Course database and Person database, on the same server, and under the control of a single, shared OSA, thereby supporting the first OSA allocation strategy as given in Section 2.1.3. The second variant separated the two remote database objects, placing each on its own independent server, thereby requiring two separate dedicated OSA/Translator pairs. Regardless of this difference, the two versions utilized the same aglet interaction.

3.3 Assumptions: OSA Database Management

The first variant of this prototype maintained the Course and Person databases on the same server, thereby insuring that any change to one database affects the other database. Utilizing a shared OSA to control the two databases simplified the testing of the system. Based on our success with the first variant, we went forward to adopt a more real-world approach in the second variant, maintaining the two databases on separate servers. This design requires dedicated OSA/Translator pairs to control and manage databases on discrete servers. Associated dissimilar databases often are maintained on discrete systems yet the integrity of their contents requires managed manipulations. Management by the database's OSA/Translator pair (see Figure 4), removes the dependence on the IRAs correct operation, but increases the complexity of the servers' implementation. This solution would be utilized if numerous simultaneous changes were to be initiated

on the database. In our second solution, the management is accomplished by splitting the responsibility for data integrity across databases into the IRAs. IRA submits a request to either database via the OSA. Upon confirmation, from the OSA/Translator pair, a second request is forwarded to the other affected databases. Thus, a change in the Course database can be migrated to the Person database via the OSA/Translator.

3.4 Messaging for Aglet Interactions

When the user initiates a request via the CA, the Translator (see Figure 4 again) is utilized to form an aglet message. The aglet message is a composition of the user's role and parameters that comprise the request (method invocation) to be performed. From CA, the message sent out has the type "request" with an embedded message as an argument. When UA receives this message, it decodes the embedded message, whose type is "taskdescription", and forwards the message to its IRA agent. Then IRA agent obtains the context address of the OSA from that message and then autonomously dispatches itself to the node on which the OSA is located. The OSA broadcasts itself in the aglet context by setting a context property as a reference when it begins execution. Thus, when the IRA arrives in the same context, the IRA can locate which the aglet with whom data can be exchanged.

For aglet programming, a mobility listener can be established for the aglet. The mobility listener defines the actions that are going to occur when the aglet arrives a new place. A mobility listener is attached to an IRA, and is responsible for sending IRA the "asksevice" message once the IRA arrives at its destination. Thus, when the IRA receives a message with the "asksevice" type, the IRA knows that it has arrived at the destination. The "askservice" message contains the user's role, the request number (method to invoke), and all of the parameters of the request. Using this information IRA can communicate with OSA and receive a reply message. The reply message contains either an error code in the case of unauthorized or unsuccessful access, or the result (success or retrieve data) of the request. After receiving a reply for OSA, IRA dispatches itself back to the originating node (client). When IRA arrives home, another message with type "back" will be sent to IRA from its mobility listener. Using this message, IRA can awaken UA and send the reply to the request for final processing by UA before it is returned to CA. Throughout the entire processing of a request, different messages are utilized to identify the current status of IRA, which along with the mobility listener, allows IRA to move and awaken as needed.

3.5 Experimentation in Graduate Course

During the Spring 1999 semester at UConn, the aglet prototype for the baseline agent approach as shown in Figure 4 and reviewed in Sections 3.1 through 3.4, was the subject of a course project to explore extensibility and reuse of aglet software. The students were presented lectures on both role-based security concepts and agent computing models, along with an examination of the concepts presented in this paper. With this as background, the students were given the aglet software consisting of 1500 lines of Java code. While the majority of the class was familiar with Java, no one in the class had programmed using agents and aglets. For the project, the students modified the agent-based implementation to includes a new Waiting List Agent (WLA), which is an application specific agent that would be responsible for monitoring the Course database for potential openings in full courses. The WLA would be spawned by CA as an option from the GUI. Once spawned, WLA would have a limited life time that is either user defined or tied to the last day of the add-drop period. WLA would be a mobile agent, free to move to the processing node where the course database is resident. When an individual drops a course or seats are added to the course, WLA would automatically enroll the individual in the course and notify him/her by email. The students implemented two versions of WLA:

- **OSA Initiated**: In this version of WLA, whenever updates to the seats in a course occur, the OSA can alert the WLA, which in turn can add the individual to the course via a path that would proceed through UA to IRA to OSA as per the normal update process.
- **WLA Initiated**: In this version of WLA, WLA is more autonomous, and is designed to periodically poll the OSA (via the normal UA to IRA to OSA path). The polling periodicity would be user definable. The polling of OSA would ask if the course requested by CA has seats. If a positive response is received, the update will proceed per the normal process.

There is a clear tradeoff between the two versions, the first requiring the maintenance of a

list of active WLAs by OSA, to allow the WLAs to be contacted, while the second raises the potential of increased network traffic due to the periodic polling. Surprisingly, all 24 students in the class successfully designed, implemented, and demonstrated these two new agents, with 75% of the students able to send email when the course was added to a students schedule. In general, the students were able to work within the agent/aglet paradigm using their knowledge of role-based security to modify and extend the existing software.

4. Concluding Remarks and Ongoing Efforts

Organizations and stakeholders must be aware of potential solutions that are appearing on the horizon in support of constructing dependable and secure distributed and web-based applications. One such emerging technology is software agents, which allows autonomous and mobile actions to occur in support of a new, dynamic, computing paradigm. Our work in this paper has emphasized the utilization of mobile and stationary agents to define a set of architectural approaches in support of role-based security for dynamically accessing remote objects. We have presented a baseline agent approach (see Section 2.1 again) that dispatches an agent to another computing node to access a remote object. This baseline approach was extended to handle complex requests, through the addition of a hierarchy of agents (see Section 2.2 again) and a object-manager to control access to collections of remote objects (see Section 2.3 again). The three approaches were compared in Section 2.4 from various dimensions to clearly understand under which situations each approach is most useful. Our successful prototyping efforts of the baseline approach utilizing IBM's aglet Java agent system was described in Section 3. Overall, we believe that agent approaches to security are of increasing interest to organizations that are involved in web-based and distributed computing applications, and we must begin to provide apropos security solutions.

Our ongoing and future efforts in this area involve both continued prototyping and exploration of agent architectures. In Section 2, many different allocation strategies for the different components of our agent-security approaches were detailed, and the prototyping effort as given in Section 3 has only discussed two variants of the baseline approach. We are actively pursuing other variants so that we can more precisely understand the tradeoffs in the different approaches. This is likely to be the subject of a future project in a graduate course (see Section 3.5 again). The architectural variants given in Section 2 are also being examined to consider other potential variants with different component structures. Also, while we have implemented our approach using aglets, there are other Java-based agent platforms (Voyager, Concordia, etc.) that are candidates for prototyping test-beds. It would be interesting to compare and contrast the different agent approaches for supporting role-based security. Finally, one glaring omission of our work is the realization of a security policy for a distributed setting. How is security for clients defined? For servers? For legacy/COTS? The ability to define, realize, and enforce such a policy is critical to insure that agents implementing security are working within a defined and refined context.

References

[Demu97] S. Demurjian and T.C. Ting, "Towards a Definitive Paradigm for Security in Object-Oriented Systems and Applications", *Journal of Compuer Security*, Vol. 5, No. 4, 1997.

[Demu98] S. Demurjian, et al., "Software Architectural Alternatives for User Role-Based Security Policies", in *Database Security, XI: Status and Prospects*, Lin and Qian (eds.), Chapman Hall, 1998.

[Gene94] M. Genesereth and S. Ketchpel, "Software Agents", *Communications of the ACM*, Vol. 37, no. 7, July 1994.

[Hale96] J. Hale, et al., "A Framework for High Assurance Security of Distributed Objects", *Proc. of 10th IFIP WG11.3 Working Conf. on Database Security*, Cumo, Italy, July 1996.

[Hale98] J. Hale, et al., "Programmable Security for Object-Oriented Systems", *Proc. of 12th IFIP WG11.3 Working Conf. on Database Security*, Chalkidiki, Greece, July 1998.

[Karj97] G. Karjoth, et al., "A Security Model for Aglets", *IEEE Internet Computing*, Vol. 1, no. 4, July-August 1997.

[Keef88] T. Keefe, et al., "A Multilevel Security Model for Object-Oriented Systems", *Proc. of 11th Natl. Computer Security Conf.*, Oct. 1988.

[Land84] C. Landwehr, et al., "A Security Model for Military Message Systems", *ACM Trans. on Computer Systems*, Vol. 2, No. 3, Sept. 1984.

[Lang98] D. Lange and M. Oshima, *Programming and Deploying Java Mobile Agents with Aglets*, Addison-Wesley, 1998.

[Loch88] F. H. Lochovsky and C. C. Woo, "Role-Based Security in Data Base Management Systems", in *Database Security: Status and Prospects*, C. Landwehr (ed.), North-Holland, 1988.

[Sand96] R. Sandhu, et al., "Role-Based Access Control Models", *IEEE Computer*, Vol. 29, No. 2, Feb. 1996.

[Smar99] D. Smarkusky, S. Demurjian, T.C. Ting, and C. Bastarrica, "Role-Based Security and Java", in Database Security, XII: Status and Prospects, S. Jajodia (ed.), Kluwer, 1999.

[Tari98] Z. Tari, "Designing Security Agents for DOK Federated System", in *Database Security, XI: Status and Prospects*, T.Y. Lin and X. Qian (eds.), Chapman Hall, 1998.

[Agle] www.trl.ibm.co.jp/aglets

[Conc] www.metica.com/HSL/Projects/Concordia

[Odys] www.generalmagic.com/technology.technology.html

[Voya] www.objectspace.com/voyager

Policy/Modeling Session

Tuesday, July 27, 1999

Chair: John Dobson
University of Newcastle

A Secret Splitting Method for Assuring the Confidentiality of Electronic Records

Andrew Po-Jung Ho

Department of Psychiatry
Los Angeles Country Harbor-UCLA Medical Center
University of California, Los Angeles
Torrance, California, 90509

aho@cortex.bic.uci.edu
(310) 222-3111

Abstract

Large databases are increasingly used to manage sensitive data; however, they are vulnerable to abuse by insiders and intruders. We describe a novel method that protects electronic records from intruders and even the most powerful of insiders, the system administrators. Confidential data are partitioned and distributed after asymmetric encryption for management over multiple databases. Data splitting and distributed management prevent unauthorized and covert access by any single party. Confidential systems using this design can work synergistically with existing security measures and are suitable for health, genomic, and financial records.

1. Introduction

Knowledge is power and large databases together with new data mining technologies are increasingly used to advance personal services and scientific discoveries. The public, however, remains distrustful of large databases especially because of privacy concerns. This distrust delays the implementation of promising technology and leads to economic and social costs. A recent example is the opposition to Iceland's national health and genomic database [1, 2].

According to a 1993 U.S. Congress Office of Technology Assessment report, the greatest threat to privacy of stored records is by persons within the system [3]. This assessment is supported by publicly known incidents of abuse by insiders [4]. How to allow workers to perform their tasks while preventing them from abusing these privileges is the most urgent subject for the managers of these databases. It is difficult to allow insiders to have the privileges needed to perform their tasks while preventing the abuse of these same powers. This is especially true for the powerful insiders, such as system administrators, who are typically able to retrieve confidential information without detection. Although the system administrators are not commonly perceived as the group that is most likely to perpetrate abuse, this vulnerability will increasingly gain significance as the size and value of the database increase, and other vulnerabilities are better addressed: Powerful insiders will increasingly become the weakest link. An important and related threat comes from intruders who obtain the system administrator's privileges. Methods that can prevent the abuse of the system administrators' privileges will also function as the last line of defense against the intruders.

Current security measures such as access control, trail audit [5], and anonymizing/encryption offer little protection against powerful insiders. Knowledgeable insiders and intruders are commonly able to bypass the access control and trail audit measures. Insiders who have access to the codebook, encryption algorithm, or the un-anonymized/un-encrypted data entering or leaving the system can defeat the anonymizing/encryption systems. For example, if a codebook is used to replace the names of the subjects with pseudonyms, the procedure can be reversed to determine the names of the subjects. If cryptographic algorithms [such as one-way hash or asymmetric algorithms) are used, insiders with access to the cryptographic algorithms can re-identify the records through a trial encryption attack even if the decryption keys are not accessible to the attacker [6]. Furthermore, insiders typically have access to un-anonymized or un-encrypted data when the system performs the anonymizing and encryption procedure. This is a weakness even when the encryption algorithm is secured by tamper-proof hardware. It is especially significant when a single coding unit is used since the insiders at that unit have access to all secrets entering the system. An example of a system with this vulnerability is the German cancer registry that uses a central coding unit to assign pseudonyms to patient records [7]. It implements separation of duty between the coding unit and three other units. Insiders at the coding unit have access to all un-anonymized records entering this system. Similarly, an U.S. Air Force design that relies on the distribution of sensitive data over multiple units suffers the same vulnerability [8]. The Air Force design uses a front-end unit to decompose

queries into sub-queries, which are sent to individual back-end databases. The back-end databases return their output to the front-end unit where they are recombined and transmitted to the user. While this design prevents the insiders at the back-end units from discovering the secrets, the insiders at the front-end unit have access to the complete contents during both the input and output phases of the transaction.

An alternative to a central anonymizing/encryption unit is a system where the users perform the anonymizing and encryption. A useful scheme that relies on distributed anonymizing or encryption must solve the problem of key management. A useful multi-user system must also provide centralized access control and decryption key management. At the same time, the powerful insiders charged with the access control and key management must not be able to abuse their privileges. To our knowledge, no existing design can achieve these goals.

Another limitation of anonymizing/encryption databases is that obviously identifiable personal data such as names, addresses, and phone numbers cannot be maintained by the system. More importantly, many other potentially identifiable data also cannot be maintained because they can be used to infer the identity of records [9]. In some applications, the absence of these data fields is acceptable; however, the "scrubbing" of all potentially identifying data such as date and place of birth inevitably reduces the usefulness of the system. For databases that maintain longitudinal health records or genetic information, it is effectively impossible to construct a sufficiently anonymous database [2].

2. Basic Design

We describe a novel approach that uses data partitioning, asymmetric encryption, and distributed processing to limit the power of each powerful insider. We show that this method can be used to implement centralized access control and maintain identifiable personal and sensitive data within an enforceable check-and-balance system of administration.

Unlike the German cancer registry and the Air Force database described previously, each database unit in our system has access to only the data needed to perform its assigned storage and processing tasks; each unit does not have access to any other part of the submitted data. This strict, "need-to-know" data isolation is maintained by cryptography throughout the data handling procedure. Separation of duty is achieved through separate administration of each database unit. By partitioning the sensitive data and distributing their storage and retrieval over different databases, sensitive information is protected from improper access by any one powerful insider. Furthermore, once fragmented into arbitrarily small units and separated from the rest of the confidential record, all personal identifiers can be managed as any other elements of information.

Figure 1 shows an example of how the secret splitting method can be applied to a database containing personally identifiable data. In Step 1, the ***User*** inputs a query into a client program. The ***Client Program*** collects information to identify and authenticate the user (subject), the client program, the record of interest (object), and the action to be performed on the record. The Client Program divides the data to create 1) ***Identifier*** sub-packet that conveys information needed to identify and authenticate the subject, object, and the client program; 2) ***Action*** sub-packet that contains the description of the operation(s) to

be performed including any new data to be added. In Step 2, the Client Program first encrypts the Action sub-packet with a code that is decodable only by Database D2; then, the encrypted Action sub-packet is combined with the Identifier sub-packet and encrypted with a code that is decodable only by Database D1. The output is called the ***Original Request.*** In systems where there are more than 2 database units, additional sub-packets can be the nested within the Action sub-packet. A variety of encryption algorithms can be used so long as the sub-packets can only be decoded by each of the intended databases. For example, a symmetric algorithm can first be used to encrypt the sub-packet and then an asymmetric algorithm used to encrypt the symmetric key (using the public keys of Database D1 and D2 as appropriate). Cryptographic algorithms and key exchange protocols have been extensively reviewed [10].

The Client Program sends the Original Request to Database D1 where ***Database D1*** (D1) decodes the Identifier sub-packet. D1 does not have the decryption key necessary to decode the encrypted Action sub-packet; therefore, the content of the Action sub-packet is inaccessible to the system administrator of D1. D1 uses the information from the Identifier sub-packet to (1) authenticate the subject, (2) determine the access level of the subject in relation to the object, and (3) determine the pseudonym of the object from a ***Pseudonym Table***. The pseudonym is a code that uniquely represents the object. In Step 3, D1 combines the access level, pseudonym of the object, client program identifier, and the encrypted Action sub-packet to form the ***Anonymous Request.*** Anonymous Request is encrypted such that it is only decodable by Database D2.

D1 sends the Anonymous Request to Database D2 where ***Database D2*** (D2) decodes it. If the subject's access level permits the execution of the operation specified in the Action sub-packet on the specified object, D2 performs the operation and returns the results, if any, to the Client Program. The results can be encrypted for transmission to the Client Program using the public key of the Client Program, the User, or a special Session Key transmitted as part of the Action sub-packet.

In this potential application, Database D2 does not contain demographics, personal, or other potentially personally identifiable information; these identifiers are stored in Database D1. The fragmentation of data diminishes the likelihood of a successful statistical inference attack on D2 by unauthorized insiders while permitting retrieval of the identifying information by authorized users. Thus, although D2 contains the list of all patients with the diagnosis of AIDS, even the system administrator of D2 cannot discover the names of these patients. The names of the patients can only be revealed when the pseudonyms are linked to the names stored in D1.

3. Access Trail Logging

Each database unit maintains an activity log to discourage system administrators from sending unauthorized queries to the system to try to decode the pseudonym or to perform unauthorized data access. The activity log at D1 records the time of transmission from D1 to D2 and the object of the query. The activity log at D2 records the time and object of the query and the destination to which the results were sent. When there is a question as to the integrity of the systems, a third party auditor can compare the two logs to determine whether

there are irregularities. The preferred procedure for a third party audit utilizes a procedure to transform these logs to protect the confidential identity of the subject-object pairs.

Periodic reports can also be generated by D1 to disclose the identity of all subjects who accessed a given object. These reports can be sent directly to the owners of the objects or to an entity given the responsibility of oversight. Inappropriate access of records can thereby be identified in a timely manner and all participants held accountable for their activities.

4. Design Variations

The basic design in Figure 1 shows the data flow from the Client Program to a first database D1, onward to D2, and then back to the Client Program. Other structural organizations are possible; for example, additional databases can be connected in multiple parallel and sequential layers to further vertically and/or horizontally partition the data. Various combinations and permutations of the design can be used to implement confidential systems with performance, data integrity, scalability, and security trade-offs. The issue of data integrity, for example, can be addressed with a design using redundant paths and subsets of databases that manage error correction codes. Although the protection of personal information is an important application of this technology, the same protection method can be applied to non-personal data.

5. Discussion

As the reliability and transfer rate of networks and cryptographic engines continue to increase, it will be increasingly feasible to trade performance for increased security and confidentiality. This data management approach allows the secure and confidential storage of electronic data with little additional performance overhead since encryption during transmission is already a necessary practice for transmission security. The performance overhead is approximately one additional encryption-decryption cycle in the most basic design and any additional setup time related to the use of two rather than one encryption/decryption procedures. As hardware-based cryptographic engines continue to improve, this performance penalty will continue to diminish. The storage overhead is also manageable since it is directly proportional to the number of Subjects and Objects.

The fragmentation of data into partitions has long been used to protect sensitive information. Fundamentally, cryptographic algorithms can be viewed as methods of fragmentation (and recombination) such that the original content can be recovered only if the fragments are recombined. The data fragments can be in the form of a decryption key, share of secret, or cryptotext. Existing schemes uniformly aim to produce data fragments that confer no information when taken individually [10]. Our approach uses data fragments that confer partial information when taken individually. When these fragments are distributed to multiple units, it is possible to perform local processing using the partial information. The fragmentation process prevents successful inference attack and precludes the need for centralized decryption key management. In this way, a powerful insider who controls any given fragment is able to perform the assigned tasks but unable to infer information about any other fragment.

To reduce the likelihood of improper access to the un-fragmented data, it is desirable to split the data as close to the time and place of their creation as possible. Each Client Program (Figure 1) splits sensitive data into fragments (sub-packets) where each fragment is encrypted such that only the intended database unit(s) can decode it. The data partitioning is achieved through asymmetric cryptography to eliminate the need for a secure channel for key exchange. Our approach differs from traditional distributed systems where the distributed nature of the architecture is hidden from the client program. With our design, the client program uses destination-specific information to encrypt and distribute the data fragments. Consequently, the client program in our system provides location and transaction transparency to the users of the system.

This work makes use of a simple but fundamental principle – asymmetric cryptographic algorithms allow the confidential delivery of data through insecure intermediaries without requiring a separate secure channel for key exchange. In this application, a portion of each query is "tunneled" through the set of first databases to reach the destination database without disclosing its confidential contents. In contrast with other "tunneling" protocols (such as in virtual private networking), our system requires the intermediate node(s) to be equal partners and share in the processing of the transaction. The use of intermediate nodes as data processing partners allows the system to separately protect the fragments and the links that connect them to each other across databases: a system can be designed where a database unit manages only the links between two other database units.

Our secret-splitting method differs from secret sharing schemes where each share of secret contains no information in isolation, and therefore each share of data cannot be used to respond to a query until the shares are fully reassembled [11]. In contrast, our approach permits the distributed and independent processing of the subqueries at each database unit without the need for re-assembly. In our system, re-assembly, as the original disassembly, occurs in the client program.

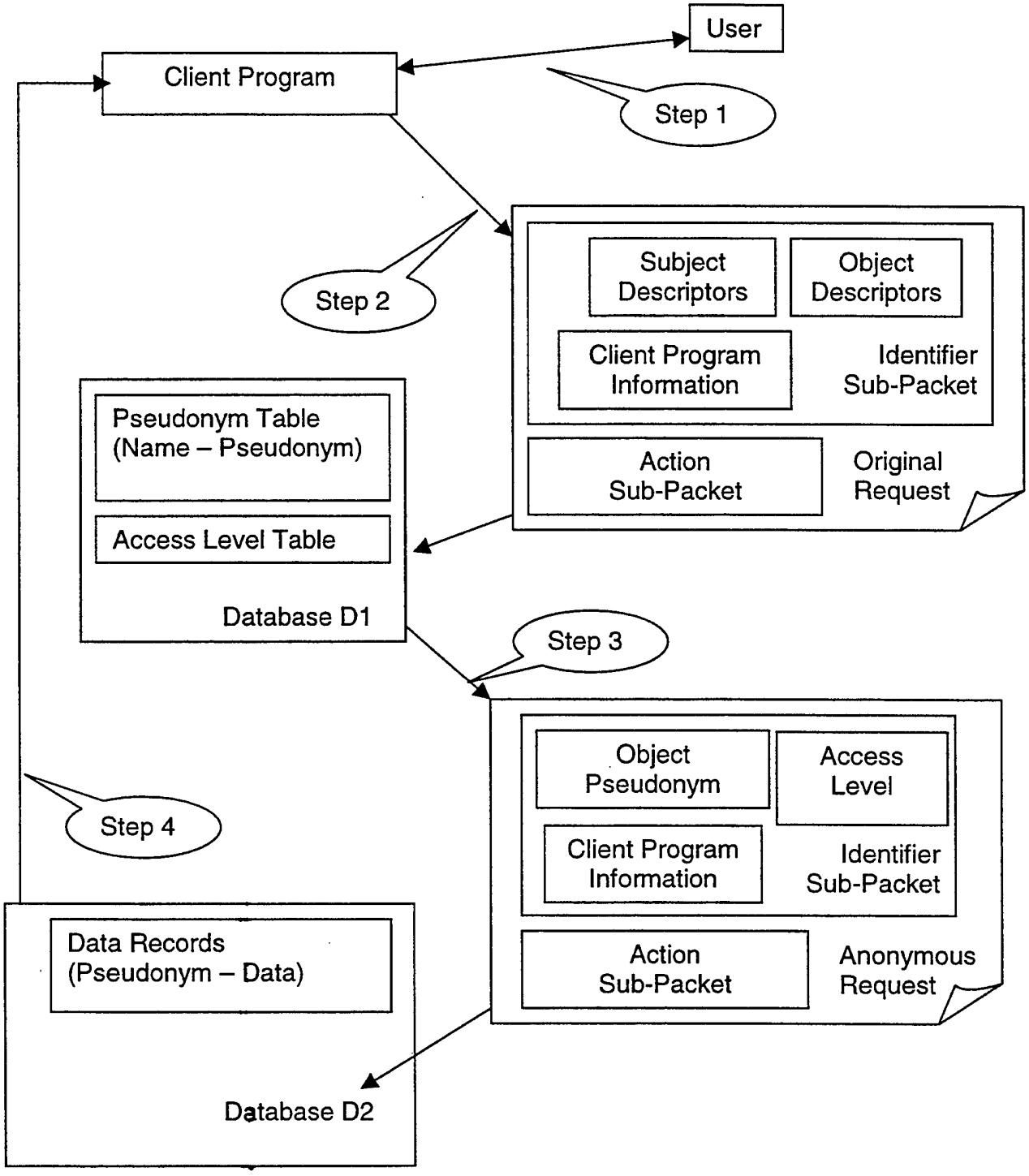
The ability to prevent abuse by powerful insiders and intruders can be instrumental in enabling the implementation of large-scale data collection and information-based applications. The secret-splitting method is potentially suited to the protection of sensitive data such as biometrics, genetic, health, and financial records. The major vulnerability of this approach comes from collusion between powerful insiders of adjacent database units. Even collusion between a user and a powerful insider will only compromise the subset of objects accessible to that user. Further analysis and modeling will be needed to characterize the relationship between variations in the design and vulnerability to collusion between units.

References

1. M. Enserink, *Science* 281, 5379 (August 14, 1998).
   M. Enserink, *Science* 281, 5380 (August 21, 1998).
   M. Enserink, *Science* 282, 5390 (October 30, 1998).
   E. Masood, *Nature* 396, 6710 (December 3, 1998).
   B. Andersen, *Science* 282, 5396 (December 11, 1998).
   M. Enserink, *Science* 283, 5398 (January 1, 1999).
   R. Haraldsdóttir, *Science* 283, 5401 (January 22, 1999).
2. R. Anderson, "The Decode proposal for an Icelandic health database", http://www.cl.cam.ac.uk/users/rja14/iceland/iceland.html (October 20,1998).
3. U.S. Congress, Office of Technology Assessment, "Protecting privacy in computerized medical information" (U.S. Government Printing Office, Washington, DC, 1993).
4. A. Zitner. *The Boston Globe* (August 1, 1997).
   Associate Press, *Chicago Tribune* (May 1997).
   Inside Healthcare Computing, March 4, 1996 (www.insideinfo.com/ security.htm).
   Inside Healthcare Computing, May 27, 1996.
5. G. Pangalos, in *Data Security for Health Care, Volume II: Technical Guidelines,* The SEISMED Consortium, Eds. (IOS Press, Washington, DC, 1996), 235-342.
6. Dusserre L, Quantin C, Bouzelat H, "A one way public key cryptosystem for the linkage of nominal files in epidemiological studies", Medinfo, 8 pt 1: 644-7, 1995.
   Bouzelat H, Quantin C, Dusserre L, "Extraction and anonymity protocol of medical file", Proc AMIA Annu Fall Symp, 323-7, 1996.
   Quantin C, Bouzelat H, Dusserre L, Med Inf, 21 (1996).
7. Michaelis J, Miller M, Pommerening K, Schmidtmann I, "A new concept to ensure data privacy and data security in cancer registries", Medinfo, **8** pt 1 (1995).
   Pommerening K, Miller M, Schmidtmann I, Michaelis J, "Pseudonyms for cancer registries", Methods Inf Med, **35** (1996).
8. J. P. O'Connor, J. W. Gray, C. McCollum, L. Notargiacomo, in *Research Directions in Database Security*, T. F. Lunt, Ed. (Springer-Verlag, New York, 1992), pp. 41-62.
9. Sweeney L, "Replacing personally-identifying information in medical records, the Scrub system", Proc AMIA Annu Fall Symp, 333-7, 1996.
   Sweeney L, "Guaranteeing anonymity when sharing medical data, the Datafly System", Proc AMIA Annu Fall Symp, 51-5, 1997.
10. B. Schneier, *Applied Cryptography, Second Edition* (John Wiley and Sons, New York, ed. 2, 1996), pp. 461-482, 561-596.
11. M. O. Rabin, *Journal of the Association for Computing Machinery* **36**, 2 (1989).

12. Received support from National Institute of Mental Health, National Research Service Award, MH14584

Figure 1. Basic Design



For Unknown Secrecies Refusal is Better than Lying

JOACHIM BISKUP
UNIVERSITÄT DORTMUND, D-44221 DORTMUND, GERMANY

A shared information system is expected to comply with the following potentially conflicting requirements. It should *provide useful answers* to arbitrary queries, while on the other hand it should *preserve certain secrets* according to a security policy. We study and compare two previously suggested approaches to meet these requirements, namely *refusal* of statements and *lying*. The investigation is performed using a highly abstract and general framework, both with respect to the information system and the preservation of secrets. The assessment shows that for unknown secrecies refusal is better than lying. In particular, while preserving the same secrets refusal can provide more useful answers.

1 Introduction

An information system is commonly employed as a tool for sharing persistent data among various users, for supporting communications among these users, and for deriving answers to queries on the basis of the stored data. In general, however, a specific user has only restricted rights to the services of the information system. In particular, query answering on behalf of a user can be restricted so as not to reveal some parts of the data which are considered to be secrets. Thus a shared information system is expected to comply with the following potentially conflicting requirements. On the one hand it should *provide useful answers* to arbitrary queries, while on the other hand it should *preserve certain secrets* according to a security policy. We study and compare two previously suggested approaches to satisfy these requirements, namely *refusal* of statements and *lying*. The investigation is performed using a highly abstract and general framework, both with respect to the information system and the preservation of secrets.

The *information system* framework basically consists of the following features. An *instance* of the information system is seen as a structure of an appropriate logic. A *query* is a sentence of that logic. And finally, a query (sentence) is *evaluated* with respect to an instance (structure) by determining whether the instance is a *model* of the query (or, in other words, whether the query is *true* in the instance).

The *preservation of secrets* framework basically consists of the following features. A *secret* might be any sentence for which the instance is a model. A security *policy* states that a given set of secrets is not allowed to be revealed to a specific user, i.e. the user should not be able to know the secrets on the basis of answers to queries. More precisely, for any instance a secret is *preserved*, if for any sequence of queries there exists an essentially different instance with the following properties: First, the system produces the same answers for the two instances, i.e., the two instances are indistinguishable for the user. Second, while the actual instance is a model of the secret, the other instance is a model of the negation of the secret.

Any approach to achieve preservation of secrets must be based on some assumptions about the *user's capabilities*. In any case these capabilities comprise the knowledge about the *system's strategy* to preserve secrets, and the system's *answers to previous queries*. Additionally, before

issuing the first query, the user has some *initial belief* about the instance which may be true or not. Thus, at any time the user's *current belief* is assumed to depend both on his initial belief and the system's answers to previous queries.

A first approach [10] to achieve preservation of secrets is based on *refusal* of statements, i.e. on just returning a special value (say `mum`), whenever the system recognizes that otherwise a secret would be challenged. A second approach [1] is based on *lying*. Deviating from [1], in this paper we use a version of lying that *always* requires to return the negation of the right answer if a challenge is detected. This version appears to be more appropriate within our context. We study both approaches in a *unified framework*, in order to assess and to compare their advantages and disadvantages. The framework is set up in the tradition of previous work on information flow control and inference control, as presented for example in [2, 3, 5, 6, 8, 4].

Our investigations focus on the case that the user supposedly does not know the secrecies, i.e. the alternatives of a sentence and its negation one of which is the right answer to be kept secret. The *insight* gained in this paper can be summarized by the statement given as the title of the paper:

- For unknown secrecies, refusal is better than lying.
- In particular, while preserving the same secrets refusal can provide more useful answers.

Formally, the insight only applies for our general framework. This framework is general in the sense that many reasonable and sufficiently powerful practical systems are expected to be specialized examples. Thus, the insight widely remains valid also for practical situations. Basically, there appears to be only two ways to overcome the results. First, we could consider less powerful query facilities which do not allow to submit any single atomic ground sentence as a query. In that case we would have more alternatives than just refusal and the simple form of lying by returning the negation of the right answer. Second, we could relax our notion of preserving a secret by only requiring that its actual computation is unlikely to be feasible within a resonable amount of time. Both ways are beyond the scope of the present paper.

2 A simple example

This section is intended to serve two purposes. It exemplifies the high level ideas and claims surveyed in the introduction. And it illustrates the formal concepts and results presented in the rest of the paper.

We assume that the underlying logic of our example information system includes the atomic sentences **a**, **b**, **s1** and **s2**, which can be used like propositional variables. Suppose the actual *instance* of the information system is given as an Herbrand structure by

$db := \{\mathbf{a}, \mathbf{b}, \mathbf{s1}, \mathbf{s2}\}$.

Furthermore, suppose that the security policy requires not to reveal whether **s1** or $\neg$**s1** holds and whether **s2** or $\neg$**s2** holds. Later on this requirement will be formally represented by the set of so-called *secrecies*, i.e. by

$secrecies := \{ \{\mathbf{s1}, \neg\mathbf{s1}\}, \{\mathbf{s2}, \neg\mathbf{s2}\} \}$.

The actual *secrets* are given by the alternatives that are valid in the instance *db*, i.e. by

$secrets := \{\mathbf{s1}, \mathbf{s2}\}$.

We will consider two different runs of controlled query evaluation under both the refusal approach and the lying approach. In both runs the user submits the sequence **a**, **b**, $\neg$**s2** of *queries* about literals, after having got a confirmation about his initial belief. The *initial believes* are chosen such that they are valid with respect to the actual instance and do not violate the security policy by themselves. Thus for all cases, when the initial belief is queried, then the ordinary query result is returned.

query	ordinary query result	answer with refusal	answer with lying	comment
b$\Longrightarrow$**s1**	**b**$\Longrightarrow$**s1**	**b**$\Longrightarrow$**s1**	**b**$\Longrightarrow$**s1**	the assumed initial belief is confirmed since no secret is challenged
a	**a**	**a**	**a**	no secret is challenged
b	**b**	mum	$\neg$**b**	otherwise the secret **s1** could be inferred using the initial belief
$\neg$**s2**	**s2**	mum	$\neg$**s2**	querying an alternative of a secrecy always results in refusal or lying, resp.

Table 2.1 Controlled query evaluation with initial belief **b**$\Longrightarrow$**s1**

Table 2.1 shows the first run, which is for the initial belief $\mathbf{b} \Rightarrow \mathbf{s1}$. The valid literal **a** is not at all related to the secrets, and thus the system can return the ordinary query result. The literal **b**, however, is just the premise of the initial belief the conclusion of which, **s1**, is one of the secrets. So the ordinary query result must be modified by refusal and lying, respectively. Finally, the literal $\neg$**s2** is not valid, and thus the ordinary query result is the unnegated alternative, i.e. **s2**. Surely, this result must be modified too, since it is a secret.

query	ordinary query result	answer with refusal	answer with lying	comment
$\mathbf{b} \Rightarrow \mathbf{s1} \vee \mathbf{s2}$	$\mathbf{b} \Rightarrow \mathbf{s1} \vee \mathbf{s2}$	$\mathbf{b} \Rightarrow \mathbf{s1} \vee \mathbf{s2}$	$\mathbf{b} \Rightarrow \mathbf{s1} \vee \mathbf{s2}$	the assumed initial belief is confirmed since no secret is challenged
a	**a**	**a**	**a**	no secret is challenged
b	**b**	**b**	$\neg$**b**	the disjunction of secrets (which is not explicitly declared to be secret) can be revealed under refusal but not under lying
$\neg$**s2**	**s2**	mum	$\neg$**s2**	querying an alternative of a secrecy always results in refusal or lying, resp.

Table 2.2 Controlled query evaluation with initial belief **b**$\Longrightarrow$**s1** $\vee$ **s2**

Table 2.2 shows the second run, which is for the initial belief $\mathbf{b} \Rightarrow \mathbf{s1} \vee \mathbf{s2}$. The valid literal **a** is treated as before. Now the literal **b** is the premise of the initial belief the conclusion of which, $\mathbf{s1} \vee \mathbf{s2}$, is the disjunction of the secrets. For refusal, at this point of time, there is no need for modification since this disjunction has not explicitly declared to be secret. If later on the disjuncts happen to be queried individually then the system can still refuse the ordinary query results. For lying, however, the system has to proceed more cau-

tiously: once the user believes that the disjunction of the secrets is valid, the system cannot avoid that later on the user can obtain a valid belief on a secret. Hence the system reacts by lying. Finally, as in the first run, $\neg$**s2** is not valid, and thus **s2** is the ordinary query result. Surely, this result must be modified too since it is a secret. Fortunately, the lie about **s2** is consistent with the previous lie about the literal **b**.

Comparing refusal and lying for the example we can already make some observations. The formal investigations confirm that they also hold in general. First, whenever the lying approach delivers the ordinary query result, so does the refusal approach. Second, sometimes refusal allows to return the ordinary result but lying does not. Third, in general both approaches have to postulate that the user does not know the secrecies. For (our version of) lying, this remark is obvious; for refusal, it can be justified as follows. In the first run, the system refuses a statement about the literal **b**. Seeing the answer mum, the user may examine why the system reacts in this way. If the user knew that the premise of his belief, **b**, is not protected but the conclusion, **s1**, he could conclude that **b** must be valid. Otherwise, there would be no reason to refuse the ordinary query result. It will turn out that the refusal approach can be adapted to the case that the user knows the secrecies, at the price of additional refusals.

3 Controlled evaluation of queries

This section introduces our framework for information systems and preservation of secrets.

3.1 Ordinary query evaluation

An *information system* maintains two kinds of data: A *schema DS* captures the universe of discourse for the intended application and is formally defined as the set of all allowed instances. An *instance db* is a *structure* which *interprets* the symbols of some logic, i.e. of the universe of discourse (see e.g. [9]). We only consider the most elementary kind of *query*, namely a sentence in the language of the logic. Given a structure *db* (stored as an instance) and a sentence Φ (issued as a query), Φ is either *true* (*valid*) or *false* in *db*, or in other words, the structure is either a *model* of the sentence or not. When a user issues a query Φ against the schema *DS*, the (ordinary) *query evaluation* $eval(\Phi)$ determines the pertinent case for the current instance *db*. Thus we formally define

$eval(\Phi) : DS \rightarrow \{true, false\}$ with $eval(\Phi)(db) := db$ `model_of` Φ,

where the boolean operator `model_of` is assumed to be appropriately specified for the logic under consideration. We also use an equivalent formalization where either the queried sentence or its negation is returned:

$eval^*(\Phi) : DS \rightarrow \{\Phi, \neg\Phi\}$ with $eval^*(\Phi)(db) :=$ `if` db `model_of` Φ `then` Φ `else` $\neg\Phi$.

Here as well as in other places we always follow the convention that a sequence of two negation symbols is automatically discarded. Our definition trivially implies that for all instances *db* and for all queries Φ we have db `model_of` $eval^*(\Phi)(db)$.

We also define the semantic relationship `implies` for *logical implication* in a standard way:
Φ `implies` Ψ :iff for every structure *db* with db `model_of` Φ we also have db `model_of` Ψ.
The complementary relationship is denoted with `not_implies`.

3.2 Inference control and modification of query results

The purpose of the *inference control* is to ensure that an ordinary query result is actually shown to the user only for those cases for which this is not considered to be *forbidden* according to some *security policy*. The parameter of the *security policy* is declared as a set *secr* of *secrecies*, each of which consists of a pair of complementary sentences $\{\Psi, \neg\Psi\}$. Then the policy states that for every secrecy in *secr* the user should not be able to determine — whether by explicit answers or by inferences — which alternative of the secrecy is correct (with respect to the current instance). We define $\texttt{secrecy}(\Psi) := \{\Psi, \neg\Psi\}$. Each set of secrecies *secr* uniquely determines the *secrets* of any particular instance *db* as

$$secrets_{secr,db} := \{eval^*(\Psi)(db) \mid \texttt{secrecy}(\Psi) \in secr\}.$$

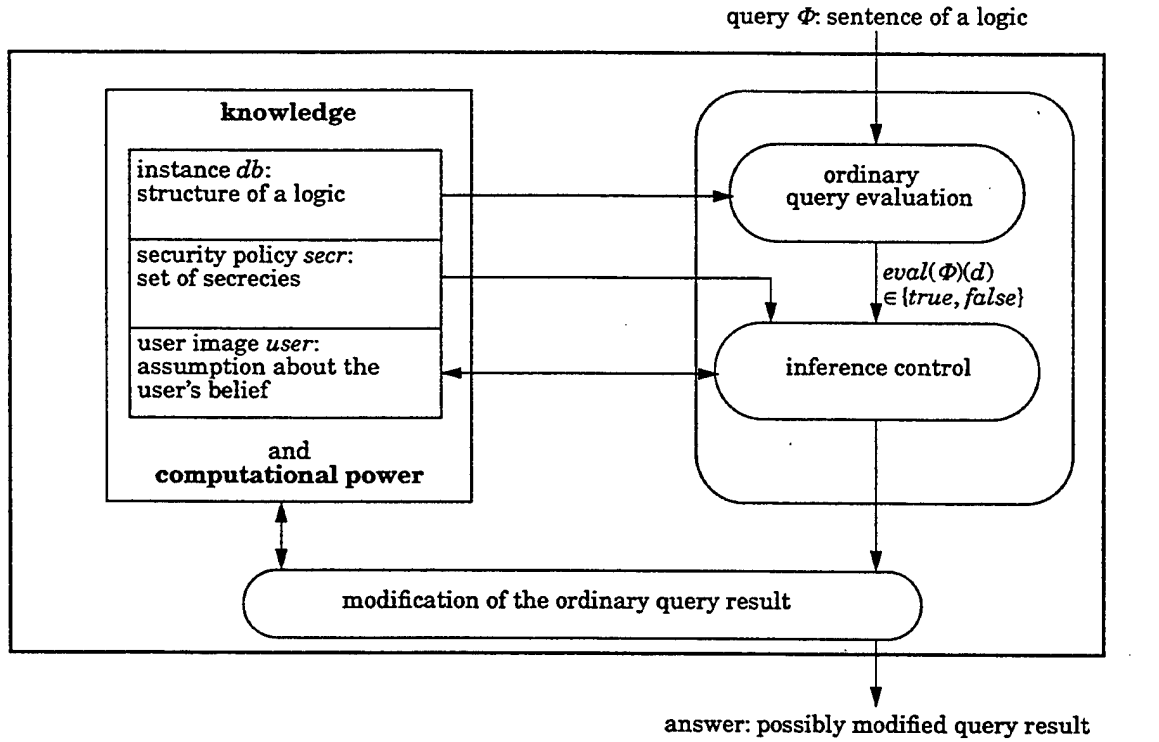


Figure 3.1 Schematic architecture of controlled query evaluation

In order to enforce any declared policy the system has to make assumptions about the user's capabilities. Since, in this scenario, the user is supposed to potentially behave as an adversary, these assumptions should be as conservative as possible, and they cannot be obtained by cooperation with that user. Our general assumptions will be as follows:

- The user knows the system's strategy of controlled query evaluation.
- Before issuing any query, the user has some *initial belief* about the instance.
- The user *remembers all answers* to previous queries.
- The user has *unrestricted computational power* to reason.

Whenever the system recognizes that the ordinary answer to a query would challenge a secret, it basically has two options to react:

- It can *refuse* any substantial statement on the query, say by returning the special value mum. Refusal has been suggested in [10]. The main results will be reviewed in Section 4.
- It can *lie* in the sense that it returns the negation of the ordinary query result. Lying has been suggested in [1] in a somehow different version that is based on secrets rather than on secrecies. The main results, suitably adopted for our version, will be restated within our framework in Section 5.

At first glance also more sophisticated forms of reactions appear to be reasonable. However, as already discussed in [10], if the system provides sufficiently powerful query facilities that allow to issue any single atomic ground sentence as a query, then, at least in principle, a clever user can always challenge the system either to refuse or to lie in the simple form.

3.2.1 The security model

Having designed any controlled query evaluation, we have to verify that the expected properties of a security model are indeed met. For our investigations the security model comprises the following features:

- The user issues *sequences* of queries.
- For any sequence of queries the controlled query evaluation should *protect the secrets.*
- The secrets should be protected in the sense that it is *in principal impossible* for the user to gain complete information about the secrets.
- *Complete information* would mean that the given query answers could result from exactly one choice for the secrets (i.e. for any secrecy $\{\Psi, \neg\Psi\}$ the pre-image of any sequence of answers should contain *at least two* essentially different instances which differ for the secrecy, one of which is a model of Ψ, and the other one is a model of $\neg\Psi$).

As sketched in Figure 3.1 the control facilities basically consist of three components:

- The first component is a *censor*, denoted by *censor*, which decides whether the ordinary query result is allowed to be shown to the user. We study censors which consider the instance *db*, the secrecies *secr*, the user image *user* and the query Φ. Thus, such a censor returns a decision of the form
  $censor(db, secr, user, \Phi) \in \{$ *true (do not show result), false (show result)*$\}$.
- The second component is a *modificator*, denoted by *modify*, which possibly modifies the ordinary query result. For refusal we will use the *refusal modificator* which constantly returns the special value mum. And for lying we will use the *lying modificator* which always returns the negated sentence.
- The third component is the *user image*, denoted by *user*, which contains the system's *explicitly* represented assumptions on the user's belief about the instance. The user image is just a set of sentences in the language of the logic. We will allow the user image to be initialized more or less freely, and for each query the returned answer is inserted but only if that answer is a sentence in the logic.

Since the controlled evaluation has to keep track of both the initial belief, $user^0$, and the previous answers, a sequence of queries $(\Phi^1,...,\Phi^n)$ is processed using a memory. Thus, for each $(\Phi^1,...,\Phi^n)$ and $user^0$ we consider a function

$control\_eval(\ (\Phi^1,...,\Phi^n), user^0\ )$

which is formalized by an inductive definition of the form:

$control\_eval(\ (\Phi^1,...,\Phi^n), user^0\ )(db, secr) := (\ (ans^1, user^1),...,(ans^n, user^n)\ )$ with

$ans^i :=$ `if` $censor(db, secr, user^{i-1}, \Phi^i)$ `then` $modify(eval^*(\Phi^i)(db))$ `else` $eval^*(\Phi^i)(db)$,

$user^i :=$ `if` ans^i is a sentence `then` $user^{i-1} \cup \{ans^i\}$ `else` $user^{i-1}$.

Any specifc approach is determined by the local functions *censor* and *modify*.

4 Refusal of statements

Controlled query evaluation with *refusal* treats queries according to the following rules:

- It delivers the ordinary query result if no secret is challenged.
- Otherwise it *refuses* to give a statement by using the refusal modificator.
- It maintains a user image *user* with "true beliefs", i.e. we always have db `model_of` *user*.
- Accordingly, for the functions $control\_eval(\ (\Phi^1,...,\Phi^n), user^0\ )$ only those arguments $(db, secr)$ are "allowed" which satisfy the precondition db `model_of` $user^0$.

Obviously we can only protect secrets which are not already initially believed. Hence we are interested in ensuring the *invariant for real secrets*:

$user^i$ `not_implies` X, for all $X \in realsecrets$, where

$realsecrets := \{\ X \mid user^0$ `not_implies` X and db `model_of` X and `secrecy`$(X) \in secr\}$.

The main results of [10] about controlled evaluation with refusal are restated in the next subsections. They can be basically summarized as follows:

- With regard to a modelled user who does *not* know the secrecies: for preserving secrets under all sequences of queries, it is necessary and sufficient that the censor maintains the invariant for real secrets. This property is just achieved by the *secret censor*.
- With regard to a modelled user who *is aware* of the secrecies: for preserving secrets under all sequences of queries, it is necessary and sufficient that the censor maintains a stronger invariant — one that protects any sentence of the secrecies under arbritrary answers. This property is just achieved by the *secrecy censor*.

4.1 The secret censor

Firstly, we define and study the *secret censor*. The secret censor demands that the ordinary query result be modified if the user could use the correct result $eval^*(\Phi)(db)$ to infer a secret Ψ, i.e.

$censor_{secrets}(db, secr, user, \Phi) :=$

(exist Ψ)[db `model_of` Ψ and `secrecy`$(\Psi) \in secr$ and

$user \cup \{\ eval^*(\Phi)(db)\ \}$ `implies` Ψ].

Theorem 4.1 below is based on the assumption, that the user does *not* know the secrecies. Theorem 4.2 below shows that this assumption is indeed crucial since otherwise the user can possibly take advantage of refusals (which are not explicitly represented in the user image).

Theorem 4.1 [succesful refusal when secrecies are unknown]

Let *censor* be the secret censor and *modify* be the refusal modificator, $(\Phi^1,...,\Phi^n)$ be an arbitrary sequence of queries and $user^0$ be an arbitrary initial belief. Then for all "allowed" arguments (*db_1*, *secr_1*) and for all real secrets X with $user^0$ `not_implies` X and *db_1* `model_of` X and `secrecy`$(X) \in$ *secr_1*, there exists a different "allowed" argument (*db_2*, *secr_2*) with the following properties:

(a) [same answers]
$control\_eval((\Phi^1,...,\Phi^n), user^0)(db\_1,secr\_1) = control\_eval((\Phi^1,...,\Phi^n), user^0)(db\_2,secr\_2)$;

(b) [different secrets] $eval^*(X)(db\_1) \neq eval^*(X)(db\_2)$.

Proof

Basically, this theorem is Theorem 10.3 from [10]. So we only sketch the proof. According to the assumption, the real secret X has the properties

$$db\_1 \texttt{ model\_of } X \quad \text{and} \tag{4.1}$$

$$user^0 \texttt{ not\_implies } X. \tag{4.2}$$

Since the secret censor maintains the invariant for real secrets, we also have for the final belief that

$$user\_1^n \texttt{ not\_implies } X. \tag{4.3}$$

Then, by the usual definition of `implies`, there exists a structure *db_2* such that

$$db\_2 \texttt{ model\_of } user\_1^n \quad \text{and} \tag{4.4}$$

$$db\_2 \texttt{ model\_of } \neg X. \tag{4.5}$$

We take *db_2* as the new instance, and we define the new secrecies as

$$secr\_2 := \{\texttt{secrecy}(\, eval^*(\Phi^i)(db\_2) \wedge user\_1^{i-1}) \mid censor_{secrets}(db\_1, secr\_1, user\_1^{i-1}, \Phi^i)\,\}, \tag{4.6}$$

where "$\wedge$ $user\_1^{i-1}$" abbreviates the conjunctive addition of the conjunction of all elements of $user\_1^{i-1}$. The new argument (*db_2*, *secr_2*) has the properties stated in the theorem:

- The argument (*db_2*, *secr_2*) is "allowed" by (4.4) and since $user^0 \subseteq user\_1^n$.
- Property (b) results from (4.1) on the one side, yielding $eval^*(X)(db\_1) = X$, and (4.5) on the other side, yielding $eval^*(X)(db\_2) = \neg X$.
- Property (a) can be verified by an induction on the sequence number i of the query.

□

In order to give an example how in the proof of Theorem 4.1 the different argument (*db_2*, *secr_2*) is actually constructed we reconsider the simple example displayed in Table 2.1. Using the notations of the theorem we start with $user^0$:= {**b⇒s1**}, *db_1* := {**a**, **b**, **s1**, **s2**},

secr_1 := { {**s1**, ¬**s1**}, {**s2**, ¬**s2**} } and X := **s1**. When processing the query sequence **a**, **b**, ¬**s2** the system generates the user images $user^1$:= {**b**⇒**s1**, **a**}, $user^2$:= {**b**⇒**s1**, **a**} and $user^3$:= {**b**⇒**s1**, **a**}. Then, for example, we can take *db_2* := {**a**, ¬**b**, ¬**s1**, ¬**s2**} as the new instance. Finally, the new secrecies are defined as

secr_2 := { secrecy(¬**b** ∧ **b**⇒**s1** ∧ **a**), secrecy(¬**s2** ∧ **b**⇒**s1** ∧ **a**) } .

The resulting new secrets are

secrets_2 := { ¬**b** ∧ **b**⇒**s1** ∧ **a**, ¬**s2** ∧ **b**⇒**s1** ∧ **a** } .

When under the same initial belief the same query sequence is evaluated with respect to the new argument (*db_2*, *secr_2*) then we indeed get the same answers. For the first query, **a**, the ordinary query result, **a**, is returned, since {**b**⇒**s1**, **a**} does not imply any new secret. For the second query, **b**, the statement is refused, since the ordinary query result, ¬**b**, together with the current user image, {**b**⇒**s1**, **a**}, would reveal the first new secret. Similarly, the statement is refused for the third query, ¬**s2**.

Theorem 4.2

Let *censor* be the secret censor and *modify* be the refusal modificator. Then there exists a sequence of queries (Φ^1,...,Φ^n), an initial belief $user^0$ and a set *secr* of secrecies with the following properties:

there exists a sequence of query answers such that its pre-image for the function *control_eval*((Φ^1,...,Φ^n), $user^0$) with respect to instances only — fixing the secrecies — contains *exactly one* element (up to query equivalence of instances).

Proof

Basically, this theorem is implicitly stated in Section 4 of [10]. ❑

The theorems also show that, at least in general, the user cannot determine initially unknown secrecies from the answers, since otherwise he could first do so and then use a reasoning like in the (omitted) proof of Theorem 4.2 to determine also the secrets. But this method would contradict Theorem 4.1.

4.2 The secrecy censor

Secondly, we define and study the *secrecy censor*. The secrecy censor demands that the ordinary query result be modified if the user could use an arbitrary answer to infer a secret or its negation, i.e.

$censor_{secrecies}(db, secr, user, \Phi)$:=
(exist Ψ) [*db* model_of Ψ and secrecy(Ψ) ∈ *secr* and
[*user* ∪ {$eval^*(\Phi)(db)$} implies Ψ or
user ∪ { ¬$eval^*(\Phi)(db)$} implies Ψ or
user ∪ { ¬$eval^*(\Phi)(db)$} implies ¬Ψ]].

Obviously the secrecy censor is stronger than the secret censor. In particular the secrecy censor maintains a strengthened *invariant for secrecies*:

- $user^i$ not_implies X, for all X ∈ *realsecrets*, and additionally if $ans^i \neq$ mum
  $user^{i-1}$ ∪ { ¬ans^i} not_implies X, for all X with secrecy(X) ∈ *secr*.

Theorem 4.3 below states that controlled evaluation with refusal based on the secrecy censor prevents a user who knows the secrecies to gain complete information about a secret. Thus the theorem also confirms that there is no need to explicitly represent refusals in the user image.

Theorem 4.3 [successful refusal when secrecies are known]

Let *censor* be the secrecy censor and *modify* be the refusal modificator, $(\Phi^1,...,\Phi^n)$ be an arbitrary sequence of queries and $user^0$ be an arbitrary initial belief. Furthermore let *secr* be a fixed set of secrecies. Then for all instances *db_1* such that (*db_1*, *secr*) is an "allowed" argument and for all real secrets X with $user^0$ `not_implies` X and *db_1* `model_of` X and `secrecy`(X) $\in$ *secr*, there exists a different instance *db_2* such that (*db_2*, *secr*) is "allowed" with the following properties:

(a) [same answers]
control_eval($(\Phi^1,...,\Phi^n)$, $user^0$)(*db_1*, *secr*) = *control_eval*($(\Phi^1,...,\Phi^n)$, $user^0$)(*db_2*, *secr*) ;

(a) [different secrets] $eval^*(X)(db\_1) \neq eval^*(X)(db\_2)$.

Proof

Basically this is Theorem 10.2 from [10]. The proof is similar to the proof of Theorem 4.1. ❑

5 Lying

Controlled query evaluation with *lying* treats queries according to the following rules:

- It delivers the ordinary query result if no secret is challenged.
- Otherwise it *lies* by using the lying modificator $modify(\Phi) := \neg\Phi$.
- It maintains a user image *user* which consists of a belief, i.e. a set of sentences for which the instance *db* might be a model or not.
- However, the function *control_eval*($(\Phi^1,...,\Phi^n)$, $user^0$) will be considered only for consistent *initial beliefs* $user^0$. It appears to be intuitively unreasonble to assume that a user believes an inconsistent set of sentences, and each such set would have the formal porperty that it implies any sentence and hence also all secrets.

Surely, controlled query evaluation should maintain *consistency of user belief* as an invariant. However, this requirements cannot always be achieved if for any query a pertinent though possibly lied statement is returned. The following example characterizes the situation. Define *secrets* := $\{\Psi_1,...,\Psi_k\}$ and $user^0 := \{\Psi_1 \vee ... \vee \Psi_k\}$. If the user issues the sequence of queries $\Psi_1,...,\Psi_k$, then the ordinary query results, namely Ψ_i, must be modified, i.e. the lying modificator generates the answers $\neg\Psi_i$. Hence we finally get the user belief $user^k := \{\Psi_1 \vee ... \vee \Psi_k, \neg\Psi_1,...,\neg\Psi_k\}$, which clearly is inconsistent.

The final event, that the user belief becomes inconsistent after k answers, is closely related to the directly preceding event, that the still consistent user belief after k-1 answers logically implies the last secret. For consider that user belief $user^{k-1} := \{\Psi_1 \vee ... \vee \Psi_k, \neg\Psi_1,...,\neg\Psi_{k-1}\}$: clearly we have $user^{k-1}$ `implies` Ψ_k. Hence the initial belief and the first k-1 answers (lies) would reveal the last secret Ψ_k.

The example indicates that in general it is necessary to maintain a strengthened *invariant* that not only preserves the consistency of the user belief but also the assertion that the user belief does not imply the *disjunction of all secrets* (which clearly includes the consistency).

The more precise investigations below will provide a main result that can be basically summarized as follows:

- With regard to a modelled user who does *not* know the secrecies: for preserving secrets under all sequences of queries, it is necessary and sufficient that the initial belief does not imply the disjunction of all secrets and that the censor maintains the corresponding invariant. This property is just achieved by the *secret-disjunction censor.*

5.1 The secret-disjunction censor

The *secret-disjunction censor* demands that the ordinary query result be modified if the user could use the correct result to infer the *disjunction of all secrets*, i.e. with

$secret\_disjunction := \Psi_1 \vee ... \vee \Psi_k$ for $secrets := \{ \Psi_1, ..., \Psi_k \}$, in particular

$secret\_disjunction := \Diamond$ for $secrets := \varnothing$,

define

$censor_{secdisj}(db, secr, user, \Phi) := user \cup \{ eval^*(\Phi)(db) \}$ `implies` $secret\_disjunction.$

Then, together with the *lying modificator* $modify(\Phi) := \neg\Phi$, by instantiating the generic declaration given in Section 3.2.1, we get *controlled query evaluation with lying* as

$control\_eval((\Phi^1, ..., \Phi^n), user^0)(db, secr) := ((ans^1, user^1), ..., (ans^n, user^n))$ with

ans^i := `if` $censor_{secdisj}(db, secr, user^{i-1}, \Phi^i)$ `then` $\neg eval^*(\Phi^i)(db)$ `else` $eval^*(\Phi^i)(db)$,

$user^i := user^{i-1} \cup \{ ans^i \}$.

Theorem 5.2 below describes the situation when controlled query evaluation with lying achieves the required kind of security. In preparation for this result, Theorem 5.1 below first shows that the *necessary* invariant for the disjunction of all secrets is indeed maintained. Both theorems together say that the invariant is also *sufficient* for preserving the secrets. These theorems are related to the assertions in Section V of [1], which were made there within a model-theoretic framework for a modal logic of belief and for a version of lying based on secrets rather than on secrecies.

Theorem 5.1 [Invariant of the secret-disjunction censor]

Let *censor* be the secret disjunction censor and *modify* be the lying modificator. Then controlled query evaluation with lying has the following property:

$user^{i-1}$ `not_implies` $secret\_disjunction$

if and only if

$user^i$ `not_implies` $secret\_disjunction.$

Proof

"⇐": The claim immediately follows from $user^{i-1} \subseteq user^i$.

"⇒": Assume

$user^{i-1}$ `not_implies` $secret\_disjunction,$ (5.1)

and let Φ^i be an arbitrary query. According to the definition of the controlled query evaluation with lying, the returned answer ans^i is either $eval^*(\Phi^i)(db)$ or $\neg eval^*(\Phi^i)(db)$, and this answer is added to $user^{i-1}$.

Case 1: $ans^i = eval^*(\Phi^i)(db)$.

This case happens if and only if the secret-disjunction censor delivers *false*, i.e. if $user^{i-1} \cup \{ eval^*(\Phi)(db) \}$ `not_implies` *secret_disjunction*. The latter relationship is just the claim.

Case 2: $ans^i = \neg eval^*(\Phi^i)(db)$.

This case happens if and only if the secret-disjunction censor delivers *true*, i.e. if

$$user^{i-1} \cup \{ eval^*(\Phi^i)(db) \} \texttt{ implies } secret\_disjunction \tag{5.2}$$

Now suppose indirectly that we also have

$$user^{i-1} \cup \{ \neg eval^*(\Phi^i)(db) \} \texttt{ implies } secret\_disjunction. \tag{5.3}$$

Then, by a well-known result of logic, (5.2) and (5.3) are only possible if even

$$user^{i-1} \texttt{ implies } secret\_disjunction. \tag{5.4}$$

This relationship, however, contradicts assumption (5.1). ❑

Theorem 5.2 [successful lying when secrecies are unknown]

Let *censor* be the secret-disjunction censor and *modify* be the lying modificator, $(\Phi^1,...,\Phi^n)$ be an arbitrary sequence of queries and $user^0$ be an arbitrary initial belief. Then for all arguments (*db_1*, *secr_1*) with $user^0$ `not_implies` *secret_disjunction_1* and for all secrets X with *db_1* `model_of` X und `secrecy`$(X) \in$ *secr_1*, there exists a different argument (*db_2*, *secr_2*) with the following properties:

(a) [same answers]
$control\_eval((\Phi^1,...,\Phi^n), user^0)(db\_1,secr\_1) = control\_eval((\Phi^1,...,\Phi^n), user^0)(db\_2,secr\_2)$;

(b) [different secrets] $eval^*(X)(db\_1) \neq eval^*(X)(db\_2)$.

Proof

Setting $user\_1^0 = user^0$ we have by assumption

$$user\_1^0 \texttt{ not\_implies } secret\_disjunction\_1 \tag{5.5}$$

and hence, by Theorem 5.1 about the invariant, also

$$user\_1^n \texttt{ not\_implies } secret\_disjunction\_1. \tag{5.6}$$

Since $X \in$ *secrets_1* and thus X `implies` *secret_disjunction_1*, from (5.6) we obtain

$$user\_1^n \texttt{ not\_implies } X. \tag{5.7}$$

Then, by the usual definition of `implies`, there exists a structure *db_2* such that

$$db\_2 \texttt{ model\_of } user\_1^n, \tag{5.8}$$

$$db\_2 \texttt{ model\_of } \neg X. \tag{5.9}$$

We take *db_2* as the new instance, and we define the new secrecies as

$$secr\_2 := \emptyset. \tag{5.10}$$

Hence we have

$$secret\_disjunction\_2 := \Diamond. \tag{5.11}$$

Then we show that the new argument (*db_2, secr_2*) has the properties stated in the theorem:

- Property (b) results from the assumption *db_1* `model_of` X on the one side, yielding $eval^*(X)(db\_1) = X$, and from (5.9) on the other side, yielding $eval^*(X)(db\_2) = \neg X$.
- Property (a) can be verified by an induction on the sequence number i of the query.

❑

In order to give an example how in the proof of Theorem 5.2 the different argument (*db_2*, *secr_2*) is actually constructed we reconsider the simple example displayed in Table 2.1. We start with $user^0 := \{\mathbf{b} \Rightarrow \mathbf{s1}\}$, $db\_1 := \{\mathbf{a}, \mathbf{b}, \mathbf{s1}, \mathbf{s2}\}$, $secr\_1 := \{ \{\mathbf{s1}, \neg\mathbf{s1}\}, \{\mathbf{s2}, \neg\mathbf{s2}\} \}$ and $X := \mathbf{s1}$. When processing the query sequence $\mathbf{a}, \mathbf{b}, \neg\mathbf{s2}$ the system generates the user images $user^1 := \{\mathbf{b} \Rightarrow \mathbf{s1}, \mathbf{a}\}$, $user^2 := \{\mathbf{b} \Rightarrow \mathbf{s1}, \mathbf{a}, \neg\mathbf{b}\}$ and $user^3 := \{\mathbf{b} \Rightarrow \mathbf{s1}, \mathbf{a}, \neg\mathbf{b}, \neg\mathbf{s2}\}$. Then we have to take $db\_2 := \{\mathbf{a}, \neg\mathbf{b}, \neg\mathbf{s1}, \neg\mathbf{s2}\}$ as the new instance. Finally, the new secrecies are defined as $secr\_2 := \emptyset$. Since the new argument has no secrets, the system will always return the ordinary query result. Accordingly, when under the same initial belief the same query sequence is evaluated with respect to the new argument (*db_2*, *secr_2*) then we indeed get the same answers.

6 Assessment of refusal and lying

In Table 6.1 we summarize the properties of the approaches to controlled query evaluation as treated in this paper. Before we start the final assessment, we emphasize two features. First, the controls should preserve the secrets *uniformly*, i.e. for all possible sequences of queries and user images, and for all "allowed" arguments the controls should be performed with the same security mechanisms (the censor and the modificator in particular). Preserving secrets means, here, that in *all* cases the user cannot uniquely determine the secret part of the current instance. Secondly, this property holds even if the user can employ *unrestricted* computional power. For, whatever sequence of queries is issued and whatever initial belief is available, the evaluation function turns out to be "nowhere injective with respect to the secrets", as far as it is succesful. Therefore, the following assessment applies for *uniform* controls to defend against *unrestricted* users.

Refusal using the secret censor has the following *advantages* in comparison to (our version of) lying using the secret-disjunction censor:

- Refusal is applicable for a *larger class* of normal arguments, because the condition for the secret censor is weaker than the condition for the secret-disjunction censor.
- More generally, the secret censor is weaker than the secret-disjunction censor, and thus the former *more often* shows the ordinary query result than the latter.
- By refusal, the user is *never misled* by lying.
- Refusal can be *combined* with a stronger censor, the secrecy censor, in order to cope with applications where the secrecies are supposedly known to the user.

	Refusal with secret censor	**Refusal with secrecy censor**	**Lying with secret-disjunction censor**
censor	censorship, if current user belief and (correct) ordinary query result imply a secret	censorship, if current user belief and arbitrary result imply a secret or its negation	censorship, if current user belief and (correct) ordinary query result imply the disjunction of all secrets
modificator	refusal modificator: *modify*() = mum	refusal modificator: *modify*() = mum	lying modificator: *modify*(Φ) = $\neg\Phi$
user image	user belief *user* with *db* `model_of` *user*	user belief *user* with *db* `model_of` *user*	user belief *user* with *user* consistent
pre-condition	argument allowed: *db* `model_of` $user^0$	argument allowed: *db* `model_of` $user^0$	initial user belief $user^0$ does not imply the disjunction of all secrets
exceptional case	initial user belief $user^0$ implies a secret, and thus all statements are refused: mechanism not applicable	initial user belief $user^0$ implies a secret, and thus all statement are refused: mechanism not applicable	initial user belief $user^0$ implies the disjunction of all secrets (in particular the initial belief $user^0$ is inconsistent): mechanism not applicable
normal case	initial user belief $user^0$ does *not* imply any secret	initial user belief $user^0$ does *not* imply any secret	initial user belief $user^0$ does *not* imply the disjunction of all secrets (and thus is *not* inconsistent)
invariant in the normal case	user belief *user* does *not* imply any secret	user belief *user* does *not* imply any secret or its negation, even if the last nonrefused ordinary query result is substituted by its negation	user belief *user* does *not* imply the disjunction of all secrets
secrecies known	secrets *cannot* be preserved: Theorem 4.2	secrets can be preserved: Theorem 4.3	secrets *cannot* be preserved: obvious
secrecies unknown	secrets can be preserved: Theorem 4.1	secrets can be preserved: Theorem 4.3	secrets can be preserved: Theorem 5.2

Table 6.1 Assessment of uniform control mechanisms to defend against unrestricted users

However, there also could be two disadvantages:

- The precondition of refusal is stronger than the precondition of lying. The former requires that the assumed initial belief is *compatible with the actual instance*, whereas the latter only demands its *consistency*.
- Refusal reveals the pure fact of modification (but neither the specific reasons behind the refusal nor, in particular, the secrecies) whereas, of course, lying is not indicated.

In conclusion, for the majority of applications we expect that the advantages of refusal will be ranked higher than its disadvantages. Therefore, in general we favour refusal over lying.

There are several directions for further interesting research. We only mention a few of them. First, we could relax the two features emphasized above. So we could include less uniform and thus more sophisticated mechanisms, and we could try to protect secrets based on the restricted computational power of users. Second, we could investigate how to combine refusal and lying and possibly even other types of reactions, in particular for nonuniform mechanisms to defend against restricted users. Third, we could extend the comparison for the original version of lying that assumes that the user knows that a fixed set of secrets (not secrecies) are to be kept secret. We conjecture that the original version is closely related to refusal with the secrecy censor. Fourth, we could study appropriate instantiations of modal logic, their expressiveness and the complexity of their decison problems, for reasoning about the belief of users from the point of view of the protecting system. Fifth, we could refine the investigation for more practical versions of information systems. The previous work on lying [1] and the study in [4] are examples of useful starting points for the last two suggestions. Finally, it would be worthwhile to embed the present and future work into the general theory of Reasoning about Knowledge as elaborated in [7].

Acknowledgements: I sincerely thank Piero Bonatti, David Embley, Barbara Sprick, Reind van de Riet and three anonymous reviewers for helpful comments on previous versions of this paper.

7 References

[1] Bonatti, P.A., Kraus, S., Subrahmanian, V.S., Foundations of secure deductive databases, IEEE Transactions on Knowledge and Data Engineering 7,3 (1995), pp. 406-422.

[2] Cohen, E., *Strong Dependency: A Formalism for Describing Information Transmission in Computational Systems*, Dept. of Computer Science, Carnegie Mellon University, 1976.

[3] Cohen, E., Information transmission in computational systems, Proc. 6th Symposium on Operating Systems Principles, 1977, pp. 133-139.

[4] Cuppens, F., Trouessin, G., Information flow controls vs inference controls: an integrated approach, Proc. 3rd European Symposium on Research in Computer Security, ESORICS 94, Lecture Notes in Computer Science 875, Springer, Berlin etc., 1994, pp. 447- 468.

[5] Denning, D,E., A lattice model of secure information flow, Communications of the ACM 19,5 (1976), pp. 236-243.

[6] Denning, D,E., *Cryptography and Data Security*, Addison-Wesley, Reading etc., 1982.

[7] Fagin, R., Halpern, J.Y., Moses, Y., Vardi, M.Y., *Reasoning about Knowledge*, MIT Press, Cambridge-London, 1995.

[8] Goquen, J.A., Mesequer, J., Unwinding and inference control, Proc. IEEE Symposium on Security and Privacy, 1984, Oakland, pp. 75-86.

[9] Shoenfield, J.R., *Mathematical Logic*, Addison-Wesley, Reading etc.,1967.

[10] Sicherman, G.L., de Jonge, W., van de Riet, R.P., Answering queries without revealing secrets, ACM Transactions on Database Systems 8,1 (1983), pp. 41-59.

Workflow Systems Session

Tuesday, July 27, 1999

Chair: Reind van de Riet
Free University

Extending the BFA Workflow Authorization Model to Express Weighted Voting

Savith Kandala
CygnaCom Solutions, Inc.
Suite 100 West 7927 Jones Branch Drive
McLean, Va 22102
e-mail: skandala@cygnacom.com

and

Ravi Sandhu
Laboratory for Information Security Technology (LIST)
Dept. of Information and Software Engineering, MS 4A4
George Mason University, Fairfax, Va 22030
e-mail: sandhu@isse.gmu.edu
Home page: www.list.gmu.edu

Abstract

Bertino, Ferrari and Atluri (BFA) have recently presented a model for specifying and enforcing authorization constraints for Workflow Management Systems (WFMS). The model is comprehensive and exhibits strong properties such as (1) a language to express constraints, (2) formal notions of constraint consistency and (3) algorithms for role-task and user-task assignments. In this paper, we extend the BFA model to include primitives for weighted voting. We show that the BFA model cannot express weighted voting in a straightforward manner, whereas Transaction Control Expressions (TCEs) proposed by Sandhu [5] incorporates this notion. Since, all other aspects of TCEs can be easily simulated in BFA, we believe that the notion of weighted voting is a fundamental operation which is missing in the BFA model. Although, we do not formally prove that BFA cannot simulate weighted voting, we make a strong case that this cannot be done easily or directly. We also show that the extended-BFA model retains all the strong properties of the BFA.

1. Introduction

In recent years, workflow management systems (WFMSs) have gained popularity both in research and commercial sectors. WFMSs are used to coordinate and streamline business processes of an enterprise. A workflow separates the various activities of an enterprise into well-defined tasks. The tasks in a workflow are usually carried out by users according to organizational rules relevant to the process represented by the workflow. The security requirements imposed by these workflow applications calls for suitable access control mechanisms. An access control mechanism enforces the security policy of the organization. Typically a set of authorizations express this security policy. This is carried out by performing a check against the set of authorizations to determine if a user intending to execute a given task on a specified object is actually authorized for it.

Quite often, security policies of a given organization are expressed in terms of roles within the organization rather than individual users. Roles represent organizational agents intended to perform certain job functions within the organization. Users in turn are assigned appropriate roles based on their qualifications and responsibilities. To directly represent such organizational security policies, an access control mechanism must be capable of supporting roles. Specifying authorizations on roles is not only convenient but reduces the complexity of administration of access control as the number of roles in an organization are significantly smaller than the number of users. Role-based authorization is particularly beneficial in workflow environments to facilitate dynamic load balancing when several individuals can perform a given task. As a matter of fact, many commercial applications support role-based authorizations.

Bertino, Ferrari and Atluri (BFA) have recently presented a model for specifying and enforcing authorization constraints for Workflow Management Systems (WFMS) in [1]. The model is comprehensive and exhibits strong properties such as (1) a language to express constraints, (2) formal notions of constraint consistency and (3) algorithms for role-task and user-task assignments. In this paper, we try to express a much older model called Transaction Control Expressions (TCEs) in the BFA model. TCEs were proposed by Sandhu [5], for the purpose of enforcing dynamic separation of duties constraints. The BFA model has a much wider scope than TCEs. So it is natural to ask whether or not BFA can simulate TCEs.

In spite, of the generality of BFA we show in this paper that the BFA model cannot fully express TCEs. In particular, the notion of weighted voting, which is a component of TCEs, cannot be expressed in the BFA model. Our paper does not prove this formally, but it does make a compelling case that there is no straightforward way of expressing weighted voting in BFA. So at least from a pragmatic viewpoint BFA should be extended to include weighted voting. We also show that the strong properties of the BFA model are still preserved in the extended-BFA model, which incorporates weighted voting.

The rest of the paper is organized as follows. Sections 2 and 3 give an overview of TCEs and the BFA model respectively. Section 4, shows how most aspects of TCEs can be easily expressed in the BFA model, except for weighted voting. Section 5 gives the extended-BFA model and section 6 shows that the properties of the BFA model are still retained in extended-BFA model. Section 7 gives the conclusion.

2. Transaction Control Expressions (TCEs)

In this section we describe the Transaction Control Expressions as proposed by Sandhu in [5]. The mechanism is very natural and intuitive , being close in spirit and letter to controls typically encountered in paper-based systems. In fact, it reflects the world of forms and books in the electronic media of databases.

Example 2.1
Consider a situation in which payment in the form of a check is prepared and issued by the following sequence of events.

1. A clerk prepares the check.
2. A supervisor approves the check.
3. A clerk issues the check.

We can specify some separation of duties constraints so that the users cannot perpetrate fraud in the system. One such constraint can be explicitly stated as, the users performing prepare, approve and issue transactions should all be different. So it will take collusion of two clerks and a supervisor to perpetrate fraud. This is called dynamic separation of duties since a clerk can perform steps 1 and 3 on different vouchers, but not on the same one. Static separation of duty would identify two kinds of clerk roles, say preparation_clerk and issue_clerk, where the former can only do step 1 and the latter only do step 3. Clearly dynamic separation of duties is more efficient.

The above example is expressed in TCEs as follows:

prepare • clerk;
approve • supervisor;
issue • clerk;

Each term of a transaction control expression has two parts. The first part names a transaction. A user assigned (explicitly or implicitly[1]) to the *role* specified in the second part can execute the transaction.

The term "prepare • clerk;"specifies that the prepare transaction can be executed on a check object only by a clerk. The semi-colon signifies sequential application of the terms. That is a supervisor can execute the approve transaction on a check only after a clerk has successfully executed the proceeding prepare transaction. Finally, separation of duties is further enforced by requiring that the users who execute different transactions in the transaction control expression all be distinct. As individual transactions are executed the expression gets incrementally converted to a history, for instance as follows.

prepare • *Alice*;	prepare•*Alice*;	prepare•*Alice*;
approve•supervisor;	approve•*Bob;*	approve•*Bob*;
issue•clerk;	issue•clerk;	issue•*Chris;*
(a)	(b)	(c)

[1] Descriptions of role hierarchies and implicit versus explicit role assignments are presented in [6].

The identity of the user who executes each transaction is recorded to enforce the requirement that these users be distinct. So if *Alice* attempts to issue that check after point (b) in this sequence the system rejects the attempt.

A transaction control expression thus contains a history of transactions executed on the object in the past and a potential history, which authorizes transactions that can be executed in the future. The expression begins as a constraint and ends as a complete history of the object. In a manual system identification of the users executing each transaction is achieved by signatures. In automated systems user identities must be recorded with guaranteed correctness.

Sometimes, different transactions in an object history must be executed by the same user.

Example 2.2
Consider the following scenario, a project leader initiates a requisition, a purchase order is prepared for the requisition by a clerk and approved by a purchasing manager. The purchase order then needs agreement of the same project leader, who was involved in requisition. Finally, the clerk places the order.

The following syntax was proposed to identify steps must be executed by the same user.

requisition •project_leader ↓ x;
prepare_order •clerk;
approve_order •purchasing_manager;
agree • project_leader ↓ x;
order•clerk;

The anchor symbol "↓"identifies which steps must be executed by the same individual. The "x"following it is a token for relating multiple anchors. For instance, in the above TCE if the same clerk had to prepare the purchase order and place the order then we can use the anchor symbol "↓"with a token "y"for the second and fifth terms in the above TCE.

In some cases, any authorized user can execute a transaction in an object history. We modify Example 2.1. Any authorized user assigned to the supervisor role can perform step 2. It would still take a collusion of a supervisor and a clerk to perpetrate fraud in the system. If the role hierarchy is such that the supervisor is senior to the clerk, then the supervisor can perform prepare and approve or he/she can perform approve and issue. But, the supervisor cannot perform all the three steps.

The following syntax identifies steps, which can be performed by any authorized user.

prepare • clerk;
approve • supervisor ↑;
issue • clerk;

Since it is a free step a token to identify multiple anchors is not needed. (Free steps are an extension to the TCE mechanism proposed in [5].)

We now turn the focus on a voting scheme scenario. In some cases, any authorized user can execute a transaction in an object history. We modify Example 2.1. The second step now requires that three different supervisor approve the check. The following syntax is used to express the voting scheme.

prepare • clerk;
3: approve • supervisor=1;
issue • clerk;

The colon is a voting constraint specifying 3 votes from 3 different supervisors. This notion is further extended to include weights of different role as follows:

prepare • clerk;
3: approve • manager = 2, supervisor = 1;
issue • clerk;

In this case, approve transactions with sufficient votes are required before proceeding to the next term. The number of votes required is interpreted as a lower bound. The moment 3 or more votes are obtained the next step is enabled.

3. The BFA model for workflow authorization

In this section we describe the workflow authorization model proposed by Bertino, Ferrari and Atluri in [1], which for convenience we call the BFA model. The BFA model gives a language for defining constraints on role assignment and user assignment to tasks in a workflow. By using this language, we can express conditions constraining the users or roles that can execute a task. The constraint language supports, among other functions, both static and dynamic separation of duties. Because, the number of tasks and constraints can be very large, the BFA model provides formal notions of constraint consistency and has algorithms for consistency checking. Constraints are formally expressed as clauses in a logic program. The BFA model also gives algorithms for planning role and user assignments to various tasks. The goal of these role-task and user-task planners is to generate a set of possible assignments, so that all constraints stated as part of authorization specification are satisfied. The planner is activated before the workflow execution starts to perform an initial plan. This plan can be dynamically modified during the workflow execution, to take into account specific situations, such as the unsuccessful execution (abort) of a task.

In the BFA model, as in most WFMSs, there is an assumption that a workflow consists several tasks to be executed sequentially. A task can be executed several times within the same workflow. Such an occurrence of a given task ***T*** is called an *activation* of ***T***. All activations of a task must be complete before the next task in the workflow can begin. Each task is associated with one or more roles. These roles are the only ones authorized to execute the task. In the remainder of this section **U, R, T** respectively denote the set of users, the set of roles and the set of tasks in a given workflow.

In the BFA model the workflow role specification is formally defined as follows.

Definition 3.1 (BFA Workflow Role Specification) A workflow role specification ***W*** is a list of task role specifications [TRS_1, TRS_2,... ,TRS_n], where each TRS_i is a 3-tuple (T_i,

$(RS_i, >_i)$, act_i) where $T_i \in$ **T** is a task, $RS_i \in$ **R** is the set of roles authorized to execute T_i, $>_i$, is a local role order relationship, and $act_i \in$ **N** is the number of possible activations of task T_i. The workflow tasks are sequentially executed according to the order in which they appear in the workflow role specification.

In order to provide a semantic foundation for the BFA model and to formally prove consistency, the constraints are represented as clauses in a normal logic program. The clauses in a logic program can contain negative literals in their body.

Definition 3.2 (Constraint Specification Language)

The constraint specification language was specified by defining the set of constants, variables and predicate symbols.

In the following ***C*** denotes a set of constraint identifiers.

- *Constant Symbols*: Every member of the set of users ***(U)***, set of roles ***(R)***, set of transactions ***(T)***, set of constraints **(C)**, and the set of natural numbers ***(N)***.
- *Variable Symbols:* There are five sets of variable symbols ranging over the sets **U, R, T, C** and **N** denoted as **Vu, Vr, Vt, Vc and Vn** respectively. The user terms are denoted with **UT** the set **Vu ∪ U.** Similarly, **RT, TT, CT, NT** denote the sets **Vr ∪ R, Vt ∪ T, Vc ∪ C** and **Vn ∪ N.**
- *Predicate Symbols:* The set of predicate symbols consists of five sets:
    1. A set of *specification predicates* ***SP*** expressing information concerning the specification of a workflow.
    2. A set of *execution predicates* ***EP*** capturing the effect of TCE execution
    3. A set of *planning predicates* ***PP*** expressing the restrictions imposed by the constraints on the set of users that can execute a task.
    4. A set of *comparison predicates* ***CP*** capturing Comparison operators
    5. A set of *aggregate predicates* ***AP*** capturing the aggregate operators

Appendix A has the table of all the predicates used in this paper, a complete list of all the predicates and their descriptions is given in [1].

A rule is an expression of the form:

H ← A1,... ... An, not B1,... ... not Bm, n,m>= 0

Where H, A1,... ..An and B1... ... ..Bm are atoms and not denotes negation by failure. H is the head of the rule and whereas A1,... ..An, not B1... ... .not Bm is the rule body. Rules can be expressed in the constraint specification language can be classified into a set of categories according to the predicate symbols they contain. Namely, explicit rules, assignment rules, static checking rules and integrity rules.

Appendix B has the table of all the rules mentioned above and their descriptions. The definitions of these rules and their description are given in [1].

Definition 3.3 (Constraint-Base) Let W be a workflow. The Constraint-Base associated with W (written CB (W)) consists of a set of explicit, assignment and integrity rules.

Intuitively, a CB is consistent if and only if the constraints it encodes are satisfiable. The consistency of a CB is determined by computing and analyzing its model. Details on CB consistency, consistency analysis and role-task / user-task assignment algorithms are presented in [1].

4. Expressing TCEs in BFA

In this section we show how the separation of duties constraints of TCEs can be expressed in the BFA model. We illustrate this by expressing all the TCEs mentioned in section 2 in BFA. In this section we also argue that the weighted voting scenario which was expressed in section 2 cannot be expressed in BFA. Although, this is not formally proved, we make a compelling case that there is no straightforward way of expressing weighted voting in BFA. So at least from a pragmatic viewpoint BFA should be extended to include weighted voting.

Basic TCE

Consider the following TCE presented in section 2 and the global role hierarchy in Figure1:

prepare • clerk;
approve • supervisor;
issue • clerk;

The separation of duties constraints can be enumerated as follows:

C_1: A user cannot execute approve, if he/she had successfully executed prepare.

C_2: A user cannot execute issue, if he/she had successfully executed prepare or approve.

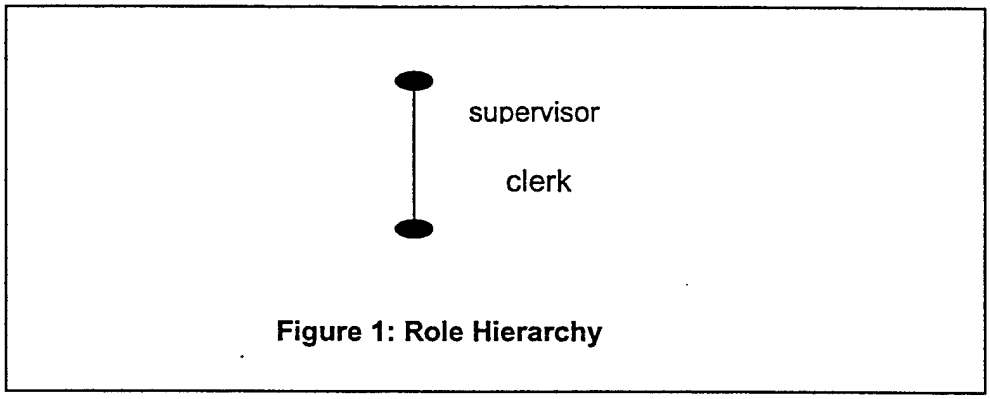


Figure 1: Role Hierarchy

These can be expressed in BFA as follows:

Workflow W = [(prepare,({clerk, supervisor}, {}), 1),
(approve,({supervisor}, {}), 1),
(issue,({clerk, supervisor}, {}), 1)]

Constraint Base CB (W):

$R_{1,1}$: $cannot\_do_u(U, approve) \leftarrow execute_u(U, prepare, 1)$;

$R_{2,1}$: $cannot\_do_u(U, issue) \leftarrow execute_u(U, approve, 1)$;

$R_{2,2}$: $cannot\_do_u(U, issue) \leftarrow execute_u(U, prepare, 1)$;

TCE with Anchors

Consider the following TCE presented in section 2 and the global role hierarchy in Figure2:

requisition •project_leader ↓ x;
prepare_order •clerk;
approve_order •purchasing_manager;
agree • project_leader ↓ x;
order•clerk;

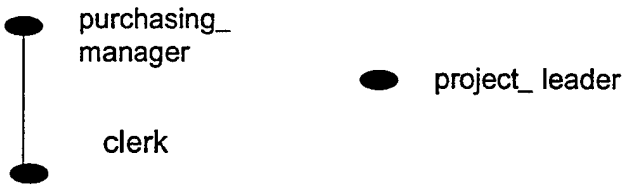


Figure 2: Role Hierarchy

The separation of duties constraints can be enumerated as follows:

C_1: A user cannot execute prepare_order, if he/she had successfully executed requisition.

C_2: A user cannot execute approve_order, if he/she had successfully executed requisition or prepare_order.

C_3: A user executing agree must be the same user who had successfully executed requisition.

C_4: A user cannot execute order, if he/she had successfully executed requisition, prepare_order, approve_order or agree.

These can be expressed in BFA as follows:

Workflow W = [(requisition,({project_leader}, {}), 1),
(prepare_order,({clerk, purchasing_manager}, {}), 1),
(approve_order,({purchasing_manager}, {}), 1),
(agree,({project_leader}, {}), 1),
(order,({clerk, purchasing_manager}, {}), 1)]

Constraint Base CB (W):

$R_{1,1}$: cannot_do$_u$(U, prepare_order) ← execute$_u$(U, requisition,1);
$R_{2,1}$: cannot_do$_u$(U, approve_order) ← execute$_u$(U, prepare_order,1);
$R_{2,2}$: cannot_do$_u$(U, approve_order) ← execute$_u$(U, requisition,1);
$R_{3,1}$: must_execute$_u$(U, agree) ← execute$_u$(U, requisition,1);
$R_{4,1}$: cannot_do$_u$(U, order) ← execute$_u$(U, agree,1);
$R_{4,2}$: cannot_do$_u$(U, order) ← execute$_u$(U, approve_order,1);
$R_{4,3}$: cannot_do$_u$(U, order) ← execute$_u$(U, prepare_order,1);
$R_{4,4}$: cannot_do$_u$(U, order) ← execute$_u$(U, requsition,1);

TCE with Weighted Voting

Consider the following TCE presented in section 2 and the role hierarchy in Figure 3. In this, TCE the notion of voting was further extended to include different weights to different roles as follows:

prepare • clerk;
3: approve • manager = 2, supervisor = 1;
issue • clerk;

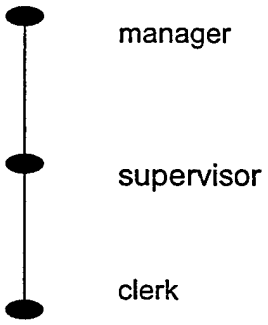


Figure 3: Role Hierarchy

In this case, the manager has a weight of 2 and the supervisor has a weight of 1 for the approve transactions. The number of votes required for approve transaction to be successful is 3. As soon as 3 or more votes are obtained the next step is enabled.

We know that the BFA model does not support assigning weights to roles (by definition 3.1). So we enumerate the various possible role assignments to approve and try to capture all the possible assignments as constraints in CB (W). Table 1 below, gives the various possible assignments for approve.

	Role Assignment for				
Possibility No.	**Activation 1**	**Activation 2**	**Activation 3**	**Total Activations**	**Votes Registered**
1	Supervisor	supervisor	Supervisor	3	3
2	Supervisor	supervisor	Manager	3	4
3	Supervisor	manager	-	2	3
4	Manger	supervisor	-	2	3
5	Manger	manager	-	2	4

Table 1: Possible role assignments for approve

Recall that, a CB for a workflow consists of a set of explicit, assignment and integrity rules (definition3.3). We now try to express all the possible role assignments for approve with these rules.

Expressing the possible role assignments as constraints using Explicit rules:
Explicit rules contain an execution or a specification atom in the head and an empty body. The various possibilities of table 1 cannot be expressed using execute $_r$ as each

TCE with Free Steps

Consider the following TCE presented in section 2 and the global role hierarchy in Figure1:

prepare • clerk;
approve • supervisor ↑;
submit • clerk;

The separation of duties constraints can be enumerated as follows:

C_1: A user cannot execute issue, if he/she had successfully executed prepare or approve.

These can be expressed in BFA as follows:

Workflow W = [(prepare,({clerk, supervisor}, {}), 1),
(approve,({supervisor}, {}), 1),
(issue,({clerk, supervisor}, {}), 1)]

Constraint Base CB (W):

$R_{1,1}$: $cannot\_do_u(U, issue) \leftarrow execute_u(U, approve,1)$;
$R_{1,2}$: $cannot\_do_u(U, issue) \leftarrow execute_u(U, prepare,1)$;

TCE with Simple Voting

Consider the following TCE presented in section 2 and the global role hierarchy in Figure1. In this TCE, equal weights are assigned to the role supervisor.

prepare • clerk;
3: approve • supervisor=1;
issue • clerk;

The separation of duties constraints can be enumerated as follows:

C_1: A user cannot execute approve, if he/she had successfully e xecuted prepare.
C_2: A user cannot execute approve, more than once. (Three different users have to execute approve).
C_3: A user cannot execute issue, if he/she had successfully executed prepare or approve.

These can be expressed in BFA as follows:

Workflow W = [(prepare,({clerk, supervisor}, {}), 1),
(approve,({supervisor}, {}), 3),
(issue,({clerk, supervisor}, {}), 1)]

Constraint Base CB(W) :

$R_{1,1}$: $cannot\_do_u(U, approve) \leftarrow execute_u(U, prepare,1)$;
$R_{2,1}$: $cannot\_do_u(U, approve) \leftarrow execute_u(U, approve,1)$;
$R_{2,2}$: $cannot\_do_u(U, approve) \leftarrow execute_u(U, approve,2)$;
$R_{3,1}$: $cannot\_do_u(U, issue) \leftarrow execute_u(U, approve,1)$;
$R_{3,2}$: $cannot\_do_u(U, issue) \leftarrow execute_u(U, approve,2)$;
$R_{3,3}$: $cannot\_do_u(U, issue) \leftarrow execute_u(U, approve,3)$;
$R_{3,4}$: $cannot\_do_u(U, issue) \leftarrow execute_u(U, prepare,1)$;

activation of approve can be executed by the supervisor or the manager. We cannot use the constraints expressed as

$execute_r$ (supervisor, approve, 1) ←

or

$execute_r$ (manager, approve, 1) ←

to capture the role assignments of table 1 since, activation 1 of approve can be executed by the supervisor or manager. So we cannot express the possible role assignments for approve in table 1 using explicit rules.

Expressing the possible role assignments as constraints using Assignment rules: Assignment rules contain a $must\_execute_u$, $must\_execute_r$, $cannot\_do_u$ or $cannot\_do_r$ atom in the head and specification atoms, execution atoms, comparison literals, or aggregate atoms in the body. The $must\_execute_r$ or $cannot\_do_r$ atoms can be used in the head of the rules to express the role assignment constraints. As each activation of approve can be assigned to a manager or a supervisor we cannot use the constraints expressed as

$cannot\_do_r$(supervisor, approve) ← $execute_r$(supervisor, approve, 1),
$execute_r$(supervisor, approve, 2) ;

or

$must\_execute_r$(supervisor, approve) ← $execute_r$(supervisor, approve, 1),
$execute_r$(supervisor, approve, 2) ;

to capture the role assignments of table 1. This is because, activation 3 of approve can be executed by the supervisor or manager, even if the activation 1 and activation 2 are successfully executed by the supervisor role. So we cannot express the possible role assignments for approve in table 1 using assignment rules.

Expressing the possible role assignments as constraints using Integrity rules: Integrity rules contain a panic atom in the head and specification atoms, execution atoms, comparison literals, or aggregate atoms in the body. The same argument presented for assignment rules holds here. So we can say, that we cannot express the possible role assignments for approve in table 1 using integrity rules.

We have already argued above, that the possible role assignments for weighted voting scenario cannot be expressed by the explicit, assignment and integrity rules of the CB. Also, the definition for a Workflow (definition 3.1) does not support assigning weights to each role. So, we conjecture there are no straightforward ways to express weighted voting schemes in BFA.

Conjecture 1: There are no straightforward ways to express weighted voting in BFA.

A stronger conjecture would assert that there are no ways, straightforward or convoluted, to express weighted voting in BFA. A formal proof of this stronger conjecture would be of theoretical interest but is outside the scope of this paper.

From a practical perspective it is quite simple to add the weighted voting feature to BFA. Weighted voting has been previously proposed in the literature and is intuitively a simple and natural concept. This is accomplished in the next section.

5. The Extended-BFA Model

In this section we describe the extended-BFA model, to accommodate the scenarios of weighted voting. We modify the definition of the BFA workflow-role specification (Definition 3.1) as follows.

Definition 5.1 (Extended-BFA Workflow Role Specification) A workflow role specification ***W*** is a list where each element in the list is either a task-role specification or a vote-role specification [TRS_1/ VRS_1, TRS_2/ VRS_2 ... ... ,TRS_n/VRS_n],
where each
TRS_i is a 3-tuple (T_i, (RS_i,$>_i$), act_i) where $T_i \in$ **T** is a task, $RS_i \in$ **R** is the set of roles authorized to execute T_i, $>_i$, is a local role order relationship, and $act_i \in$ **N** is the number of possible activations of task T_i.
and each,
VRS_i is a 4-tuple (T_i, (RS_i,$>_i$),*VotesRequired, RoleWeight*) where $T_i \in$ **T** is a task, $RS_i \in$ **R** is the set of roles authorized to execute T_i , $>_i$, is a local role order relationship, and *VotesRequired* $\in$ **N** is the number of votes required to make of task T_i and *RoleWeight* is a function that maps each role in the set RS_i to a *weight* $\in$ **N** for a given T_i.
RoleWeight: $RS_i \rightarrow$ **N**

The workflow tasks are sequentially executed according to the order in which they appear in the workflow role specification.

We propose two possible solutions for expressing the weighted voting constraints in extended-BFA model.

Solution 1 for expressing the weighted voting constraint in extended-BFA:
The weighted voting constraint could be expressed as in terms of the number of successful activations of the task *approve* as follows:

$cannot\_do_u(U, issue) \leftarrow count(success(approve, k), execute_r(R_i, approve, k),$
$R_i = manager, n_1),$
$count(success(approve, k), execute_r(R_j, approve, k),$
$R_j = supervisor, n_2),$
$(2^* n_1 + n_2) < 3;$

Here, we count the number of successful executions of the task *approve* by the role manager (returned as n_1), the number of successful executions of the task *approve* by the role supervisor (returned as n_2) and compute the votes registered as ($2^* n_1 + n_2$). Since the manager has a weight of 2 and the supervisor has a weight of 1 . If the votes registered are less than 3, then users cannot perform the task *issue*.

This solution is inefficient if there are a number of different roles assigned different weights for the task. Therefore, we propose a second solution to express weighted voting efficiently in the extended-BFA model.

Solution 2 for expressing the weighted voting constraint in extended-BFA:
We add the following weighted voting predicates and the weighted voting rule to efficiently express weighted voting in the extended BFA model.

Predicate	Arity	Arguments' Type	Meaning
role_weight	3	RT, TT, NT	role_weight(R_i, T_j, n) gets the weight of the role R_i assigned to the Task T_j and returns this value as n.
votes_required	2	TT, NT	votes_required(T_i, n) gets the minimum number of votes required for the task T_i to be successful and returns this value as n.

count_votes	2	TT, NT	count_votes(T_i, n) computes the following:
			
			for each $R_i \in RS_i$
			begin
			sum:=0
			count(success(T_i, k), execute$_r$(R_i, T_i,, k), n_1)
			role_weight(R_i, T_i, n_2)
			sum := sum + (n_1 * n_2)
			end;
			
			and returns the value of sum as n.

Table 2: Weighted Voting predicates

Definition 5.2 (Extended-BFA Constraint Specification Language)
The extended-BFA Constraint Specification Language is unchanged from definition 3.2 except for the addition of the weighted voting predicates described in table 2.

Rule	Head	Body
Weighted Voting	cannot_do$_u$	conjunction of comparison literals and weighted voting predicates

Table 3: Weighted Voting rule

Definition 5.3 (Constraint-Base) The extended-BFA Constraint-Base is unchanged from definition 3.3 except for the addition of the weighted voting rule described in table 3.

With the help of above weighted voting predicates and weighted voting rule we can now express a TCE with a voting scheme in Extended-BFA. Consider the TCE presented in section 2 and the global role hierarchy in Figure 3.

prepare • clerk;
3: approve • manager = 2, supervisor = 1;
issue • clerk;

The separation of duties constraints can be enumerated as follows:

C_1: A user cannot execute approve, if he/she had successfully executed prepare.
C_2: A user cannot execute approve, more than once. (Three different users have to execute approve).
C_3: A user cannot execute issue until approve transaction registers the required number of votes.
C_4: A user cannot execute issue, if he/she had successfully executed prepare or approve.

These can be expressed in Extended-BFA as follows:

Workflow W = [(prepare,({clerk, supervisor, manager}, {}), 1),
(approve,({supervisor, manger}, {}), 3,{(supervisor,1),(manager,2)}),
(issue,({clerk, supervisor, manager}, {}), 1)]

Constraint Base CB(W) :

$R_{1,1}$: cannot_do$_u$(U, approve) ← execute$_u$(U, prepare,1);
$R_{2,1}$: cannot_do$_u$(U, approve) ← execute$_u$(U, approve, 1);
$R_{2,2}$: cannot_do$_u$(U, approve) ← execute$_u$(U, approve, 2);
$R_{3,1}$: cannot_do$_u$(U, issue) ← count_votes(T_i, n_1), votes_required(T_i, n_2), $n_1 < n_2$;
$R_{4,1}$: cannot_do$_u$(U, issue) ← execute$_u$(U, approve, 1);
$R_{4,2}$: cannot_do$_u$(U, issue) ← execute$_u$(U, approve, 2);
$R_{4,3}$: cannot_do$_u$(U, issue) ← execute$_u$(U, approve, 3);
$R_{4,4}$: cannot_do$_u$(U, issue) ← execute$_u$(U, prepare,1);

The rule $R_{4,3}$ will only be effective if the approve task had three activations (the maximum number of possible activations). If the number of activations for the approve task was less then three, then this rule will be ineffective as execute$_u$(U, approve, 3) will always be false.

6. Properties of the Extended BFA model

We now focus the attention on showing that the properties of the BFA model are still preserved in the extended-BFA model.

The properties of the BFA model as mentioned earlier are:
(1) a language to express constraints
(2) formal notions of constraint consistency and
(3) algorithms for role-task and user-task assignments.

In the extended-BFA model, we have not changed the language to express constraints, so property (1) is preserved. The formal proof for the following proposition as presented in [1] still holds.

Proposition 6.1 *Any CB is a stratified normal program. Hence, it has a unique stable model.*

Since we have not changed the definitions of explicit, assignment and integrity rules. The formal proof presented for this proposition in [1] still holds for these rules. We now extend the argument presented in [1] to include the weighted voting rule.

A program P is stratified if its extended dependency graph does not contain any cycle involving an edge labeled with "hot"[10], where the extended dependency graph of a program P is a graph whose nodes are the predicates that appear in the heads of the rules of P. Given two nodes p1 and p2 there is a direct edge from p1 to p2 if and only if predicate p2 occurs positively or negatively in the body of a rule whose head predicate is p1. The edge (p1, p2) is marked with a "hot"sign if and only if there exists at least one rule r with head predicate p1 such that p2 occurs negatively in the body of r. By Definition 5.3, the CB associated with a given workflow consist of a set of explicit, assignment, integrity and weighted voting rules. The explicit, assignment and integrity

rules cannot form a cycle in the extended dependency graph [1]. By definition the weighted voting rule has a planning predicate (cannot_do$_u$) as head and a conjunction of weighted voting predicates and comparison literals as body. Since, the predicates that appear in the head are disjoint from the predicates that can appear in the body, they cannot form any cycle in the extended dependency graph. Hence, the extended dependency graph associated with a CB does not contain any cycle. Thus, the CB is stratified.

The static analysis algorithm and the pruning algorithm need some modifications to accommodate the VRS_is in addition to the TRS_is. These modifications are simple to make in order to preserve property (2).

The role-task and user-task assignment algorithms also need modification, to accommodate the VRS_is. Instead, of looping through the number of activations in each TRS_i they have to also consider, that each element of the workflow can be a VRS_i. In case, an element is a VRS_i. they need to keep track of the number of votes required after each assignment and the weights assigned to each role. With these modifications, property (3) of the BFA model can also be preserved.

Thus, we argue that with some modifications to the static analysis algorithm, pruning algorithm, role-task assignment algorithm and user-task assignment algorithm the extended-BFA model preserves the strong properties of the BFA model.

7. Conclusion

In this paper, we have shown that the BFA model cannot be used to express the weighted voting scenario. We have extended the BFA model so that this feature can be accommodated in the BFA model. We have also argued that the extended-BFA model does preserve all the properties of the BFA model. It should also be noted that the constraint specification language of BFA is not intended for end-users to express constraints, it is rather used internally by the system to analyze and enforce constraints. The TCEs on the other hand are very natural and intuitive, so we can use TCEs as a language in which users can specify their separation of duties constraints. The constraints in TCEs can be translated to BFA. This reduction to extended BFA also provides a formal semantics, which has so far not been given.

Future directions of our research could involve developing an automated system to translate TCEs into the BFA model, this could be helpful as the BFA model has formal semantics for expressing constraints at the system level. The BFA model and the TCEs follow a strict sequence of execution. We would also like to introduce some parallelisms in the task execution.

References

[1] Bertino. E, Ferrari. E, Atluri. V "A Flexible Model for the Specification and Enforcement of Authorization Constraints in Workflow Management System", Proceedings of the Second ACM Workshop on Role-Based Access Control, November 1997.

[2] Clark D.D, Wilson D.R "A Comparison of Commercial and Military Security Policies" Proceedings of IEEE Symposium on Security and Privacy, 1987, pg 184-194.

[3] Das S.K. "Deductive Databases and Logic Programming"Addison-Wesley, 1992.

[4] Nash M.J, Poland K. R "Some Conundrums Concerning Separation of Duty", Proceedings of IEEE Symposium on Security and Privacy, 1987, pg 201-207.

[5] Sandhu, R "Transaction Control Expressions for Separation of Duties" Proc. 4th Aerospace Computer Security Applications Conference December 1988, pages 282-286
[6] Sandhu R, Coyne E.J, Feinstein H.L, Youman C.E "Role-based Access Control Models". IEEE Computer 29(2) pg :38-47, February 1996.

[7] Sandhu R "Separation of Duties in Computerized Information Systems"Proceedings of the IFIP WG 11.3 Workshop on Database Security, September 1990.

[8] Simon R.T and Zurko M.E "Separation of Duty in Role-Based Environments" Proceedings of Computer Foundations Workshop X, June 1997.

[9] Thomas R.K, Sandhu R, "Task-based Authorization Controls (TBAC): A Family of Models for Active and Enterprise-oriented Authorization Management"Proceedings of the IFIP WG 11.3 Workshop on Database Security, August 1997.

[10] Ullman J. "Principles of Database and Knowledge-Base Systems"(2 nd Volume). Computer Science Press New York 1989

Appendix A (Predicates used in this paper)

Predicate	Arity	Arguments' Type	Meaning
$execute_u$	3	UT, TT, NT	If $execute_u$ (u_i, T_j, k) is true is true, then the k-th activation of task T_j is executed by user u_i.
$execute_r$	3	RT, TT, NT	If $execute_u$ (R_i, T_j, k) is true is true, then the k-th activation of task T_j is executed by role R_i.
success	2	TT, NT	success(T_i, K) is true if the k-th activation of task T_i within a workflow successfully executes
$must\_execute_u$	2	UT, TT	If $must\_execute_u(u_i,T_j)$ is true then task T_j must be executed by user u_i.
$must\_execute_r$	2	RT, TT	If $must\_execute_r(R_i,T_j)$ is true then task T_j must be executed by role R_i.
$cannot\_do_u$	2	UT,TT	If $cannot\_do_u(u_i, T_j)$ is true then the task T_j cannot be assigned to user u_i.
$cannot\_do_r$	2	RT,TT	If $cannot\_do_r(R_i, T_j)$ is true then the task T_j cannot be assigned to role R_i.

Appendix B (Constraint Specification Language Rules)

Rule	Head	Body
explicit	execution or specification atom	empty
assignment	$must\_execute_u$, $must\_execute_r$, $cannot\_do_u$ or $cannot\_do_r$ atom	specification, execution, or comparison literals, or aggregate atoms
static checking	statically_checked atom	specification, comparison literals, or aggregate atoms. Each literal in an aggregate atom is a specification or a comparison literal
integrity	panic atom	specification, execution, or comparison literals, or aggregate atoms
static	planning or specification atom	specification, comparison literals, or aggregate atoms. Each literal in an aggregate atom is a specification or a comparison literal
dynamic	planning execution or specification atom	specification, execution, comparison literals, or aggregate atoms. At least a literal in the rule must be an execution literal.

A Strategy for an MLS Workflow Management System

Myong H. Kang, Judith N. Froscher, Brian J. Eppinger, and Ira S. Moskowitz
Information Technology Division
Naval Research Laboratory

{mkang, froscher, eppinger, moskowitz}@itd.nrl.navy.mil

Abstract

Current DoD information systems need to support many different missions through cooperation with different organizations and allies. In today's fast paced and dynamic environment, it is almost impossible to design and implement a different information system for each mission. Therefore, DoD needs MLS workflow management systems (WFMS) to enable globally distributed users and existing applications to cooperate across classification domains to achieve mission critical goals. An MLS WFMS that allows users to program multilevel mission logic, securely coordinate widely distributed tasks, and monitor the progress of the workflow across classification domains is required. In this paper, we present requirements for MLS workflow and a strategy for implementing it, especially the method for decomposing an MLS workflow into multiple single-level workflows

1. Introduction

The streamlining of today's business processes has brought about significant increases in productivity and the ability for US companies to compete in the global market place. These re-engineering activities have as their basis an even greater reliance on information technology and automation. As a result, software developers have been challenged to streamline the software development process and to produce software that can be reused, that separates concerns, that supports autonomy and heterogeneity in a distributed computing environment, that allows extensibility, etc. The information technology (IT) community has developed distributed object computing standards, like CORBA and DCOM, that provide a basic level of interoperability among distributed applications. The next step is to build application specific software architectures that encode business logic for coordinating widely distributed manual and automated tasks in achieving enterprise level objectives. To assist business process modeling, vendors have developed several automated tools that help manage dependence among activities and users. A WFMS makes these tools available to users and allows them to monitor the business processes. This technology manages activities within a distributed computing environment comprising of loosely coupled, heterogeneous, autonomous, new (and legacy) components which may include transactional systems (i.e., DBMS). It provides a capability for defining the:

- Business logic,
- Tasks that make up business processes, and
- Control flow and data dependence among those tasks.

The potential benefits of this technology are enormous because of its broad reach to manage business process productivity. Industry is beginning to turn to workflow technology in order to minimize its manpower needs, optimize IT investment, achieve good performance, use legacy systems, gain flexibility for supporting evolving business goals, as well as to capitalize on advanced technology. However, commercial WFMS do not support distributed mission critical applications. They do not ensure mission critical properties, such as recoverability nor do they enforce access control policies, and certainly not multilevel secure (MLS) access control policies. Even though there is a great need for this technology in DoD, DoD cannot rely on commercial WFMS to protect national security information and perform mission critical business processes.

The constant aspect of the military challenge is change. The challenge is to respond to new threats in completely different environments. For example, today's military supports disaster relief, drug interdiction, peace-keeping missions in worldwide regional skirmishes, treaty enforcement, as well as the traditional role of national defense against aggression and weapons of mass destruction. At no other time in the nation's history has the military relied so heavily on IT systems for all of its operations, including command and control, logistics, surveillance and reconnaissance, personnel management, finances, etc. Challenging requirements of those systems include,

- The organizations that participate in a dynamic coalition may be located in different classification domains.
- The guidelines for sharing and exchanging information among organizations in different classification domains are stricter than those for organizations in the same classification domain.

To address those problems, the Naval Research Laboratory (NRL) has embarked upon a research project to build an MLS WFMS. The goal of the project is to develop tools and security critical components that allow enterprises to harvest emerging commercial off-the-shelf (COTS) technology and still rely on legacy resources with reduced risk.

In short, MLS WFMS should support:

- Secure interoperability among tasks that reside in different classification domains, and
- Maximum use of commercial software/hardware.

The need for a WFMS that can manage MLS workflows is immediate and imposes an implementation strategy that allows operational users to exploit advances in COTS technology and also to satisfy their mission critical requirements. The multiple single-level architecture, described in [6,7,12], provides the foundation for enforcing information flow requirements. A WFMS comprises several tools and runtime enactment services. This paper examines what properties these components must satisfy in a multiple single-level distributed computing environment. The MLS workflow design tool is of particular interest because the commercial tool must be significantly changed to represent classification domains. While this does not fit the paradigm of using unmodified COTS products with high assurance security devices, finding a research team that is developing a WFMS that supports mission critical workflows makes it possible to develop an MLS WFMS.

In this paper, we present requirements for MLS workflows, tools for supporting MLS workflows, and a strategy for implementing them. We also examine an MLS workflow model that supports MLS cooperation, and describe the decomposition of an MLS workflow into multiple single-level workflows.

2. Tools to Support MLS Workflow

A WFMS consists of, in general, three components:

- Workflow design (build-time) tool,
- Workflow enactment (runtime) service, and
- Monitoring tool.

A workflow design tool is a distributed programming tool with a graphical user interface. Users can express data dependence and control flow among tasks using this tool. Once a user specifies the mission logic (i.e., distributed programming logic), code for workflow enactment can be generated. A workflow enactment service is responsible for task scheduling, enforcing dependence among tasks, passing data from one task to another, and error recovery based on the generated code. The workflow monitoring tool, in general, tracks and monitors the progress of execution.

DoD needs all the tools that single-level WFMS provides. However, DoD requires extra capabilities in those tools to support MLS workflow. We will examine the extra requirements for each tool.

Design Tool for MLS Workflow

A workflow design tool is a distributed programming tool with a graphical user interface that provides the global picture of the whole mission. MLS workflow designers should be able to specify (1) tasks in different classification domains and (2) data and control dependence (flow) among them. Based on this workflow design, a specification for workflow runtime can be generated. Final runtime code that will be securely executed on an MLS distributed architecture is generated, based on this specification. The runtime specification and code generation processes, in general, depend on the underlying MLS distributed architecture.

In MLS applications, each subtask may be located in a different classification domain. The design tools for single-level workflow do not provide a capability to specify classification domains or compartments. In other words, the whole drawing area of the workflow design tool belongs to one classification domain. It also does not generate runtime code that can be executed on the underlying MLS distributed architecture. What is needed is a design tool for MLS workflow that:

- Allows MLS workflow designers to divide a design area into multiple domains,
- Allows MLS workflow designers to specify information flow and dependence among tasks that are in the same or different domains, and
- Allows MLS workflow designers to specify dominance relationships among domains (e.g., Top Secret > Secret > Unclassified).

For example, a workflow designer should be able to specify an MLS workflow as in Figure 1 where ovals represent tasks and arrows represent control and data flow. In the figure, B (begin), S (success), and F (failure) represent special tasks.

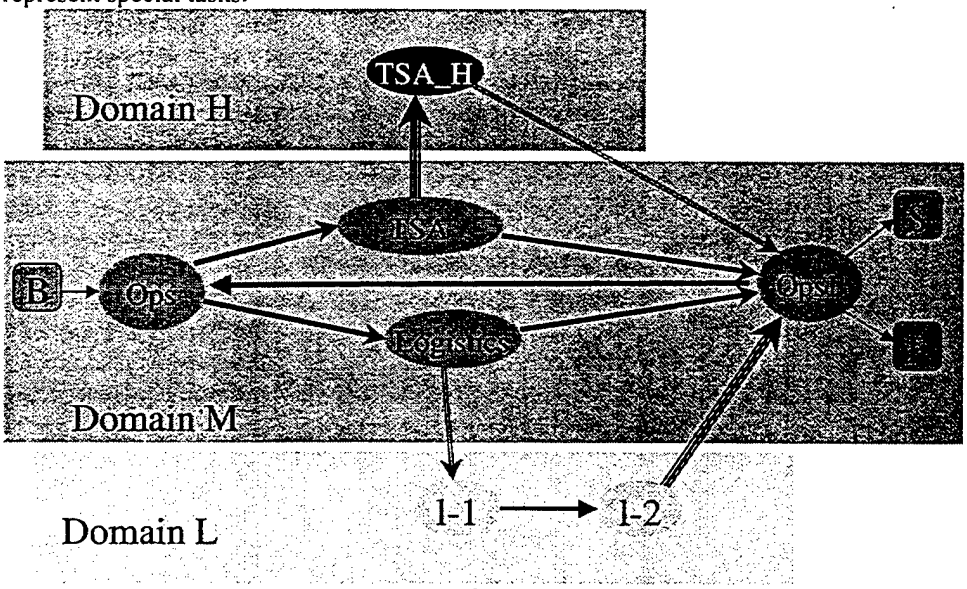


Figure 1: An Example of an MLS Workflow Design

Even though this tool allows workflow designers to specify information and control flow among tasks in different domains, the operational environment of the tool will be system-high (i.e., the workflow design tool neither accesses sensitive data in multiple domains nor passes it around). Hence, although this tool has to be trusted in the sense that it does what it is supposed to do, it can be run on a single-level system.

MLS workflow has another functional requirement. When an MLS workflow is designed, it must often interact with tasks in other domains about which the designer is not allowed to know the details. For example, when a secret workflow designer designs a workflow, the secret workflow may need to send a request to a top secret task. The secret designer is not allowed to know how the top secret task gets the

answer, but he knows how to send a request and how to receive an answer. Hence, the top-secret task, in this case, is a task foreign to the secret designer (i.e., the top secret task does not belong to his workflow). In fact, the H workflow in figure 1 may be designed in a completely different organization from the M workflow and the L workflow. A workflow design tool for MLS workflow should be equipped with the capability to model interactions with a task for which only its interface specification is known. We believe this capability to model foreign tasks has broader application, even for single-level workflows (e.g., two cooperative workflows run by two different organizations).

Enactment Service for MLS Workflow

An MLS workflow enactment service is responsible for executing a workflow in a correct and secure manner. Hence, it depends on services in the underlying MLS distributed architecture to coordinate information and control flow among tasks in different classification domains. What we need is to make secure use of single-level COTS workflow enactment services with or without modifications. As we will explain in section 3, we plan to use multiple single-level workflow enactment services to execute an MLS workflow. Since there will be no direct communication among workflow enactment services at different classification domains, there are no special MLS requirements for a workflow enactment service itself. On the other hand, the underlying MLS distributed architecture and its security devices must provide the necessary assurance for multilevel security.

Monitoring Tool for MLS Workflow

When MLS workflow is executed, there are many automatic and human computer tasks that are executed in different classification domains. Workflow managers in different classification domains (there may be a workflow manager per classification domain) may have knowledge about tasks in their classification domain and other domains they are authorized to access. In other words, users of MLS workflow in different classification domains may have different views of the workflow they are running. Hence, an MLS WFMS should provide the ability to monitor activities in all domains the workflow manager is authorized to access. Monitoring may include when, where, and who performs the tasks in the case of human tasks. Because the expected, legal behavior of a workflow is specified, the workflow monitor can be designed to detect security critical events as well as unexpected behavior. Additionally, responses for security exceptions can be specified as part of the workflow design.

3. A Strategy for MLS Workflow

An MLS WFMS should support functionality equivalent to a single-level WFMS from the perspective of MLS users who design and use multilevel workflows. Tasks that may be single-level individually but located in different classification domains, have to cooperate to achieve a higher level MLS mission.

To provide MLS services in a distributed and heterogeneous computing environment, the following information flow requirements must be enforced:

- High users must have access to low data and low resources,
- High processes must have access to low data, and
- High data must not leak to low systems or users.

An MLS WFMS should obey this MLS policy. Atluri *et. al.* investigated MLS workflow in general [13, 14]. There are two basic ways to enforce the MLS policy in MLS workflow systems:

- Build high-assurance MLS WFMS that will run on an MLS platform, or
- Build an MLS workflow by integrating multiple single-level workflows with an MLS distributed architecture.

The development of high-assurance software, necessary to provide separation between unclassified and TS/SCI information, such as MLS workflow systems, has proven to be both technically challenging and expensive. Today's fast paced advances in technology and the need to use COTS products make the traditional MLS approach untenable. Therefore, we have chosen the second approach for building MLS WFMS. It is more in line with the modern distributed computing paradigm than the first approach in terms of supporting autonomy and heterogeneity.

To implement an MLS WFMS using the architectural method, the following technical approach has been established:

- Choose an MLS distributed architecture where multiple single-level workflows can be executed.
- Choose a strategy for dividing an MLS workflow into multiple single-level workflows.
- Choose a single-level WFMS to execute single-level workflow in each classification domain.
- Implement the necessary tools for supporting MLS workflow.
- Extend the workflow interoperability model to accommodate the communication among workflows at different classification domains.
- Extend the single-level workflow enactment service to accommodate communication among tasks in different classification domains.

MLS Distributed Architecture

Composing an MLS workflow from multiple single-level workflows is the only practical way to construct a high-assurance MLS WFMS today. In this approach, the multilevel security of our MLS workflow does not depend on single-level WFMS but rather on the underlying MLS distributed architecture. Thomas and Sandhu have proposed task-based authorization for single-level workflows [15]. The MLS distributed architecture will:

- Host multiple single-level workflows to be executed and
- Provide conduits for passing information among tasks in different classification domains.

Our MLS distributed architecture is based on a security engineering philosophy: a few trusted devices in conjunction with information release and receive policy servers enforce the information flow policy of the classification domains, and single-level systems and single-level engineering solutions provide other functionality. A generic MLS distributed architecture is shown in Figure 2.

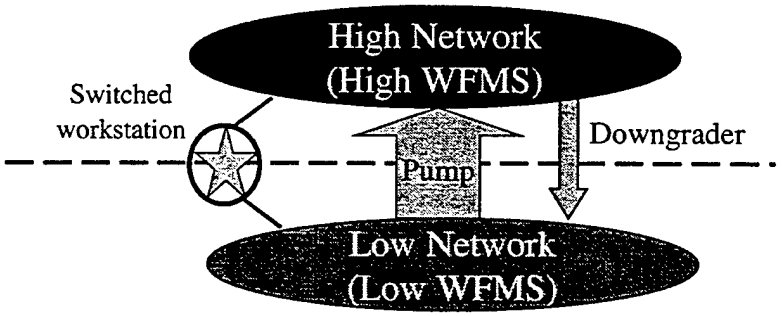


Figure 2: An MLS Distributed Architecture

In this architecture, switched workstations (e.g., "Starlight" [1]) enable a user to access resources in multiple classification domains and create information in domains that the user is authorized to access. One-way devices (e.g., a flow controller such as "A Network Pump" [5]) together with information release and receive policy servers provide a secure way to pass information from one classification domain to another. An information release policy server resides in a classification domain where the information is

released, and an information receive policy server provides a secure way to pass information from one classification domain to another as shown in Figure 3.

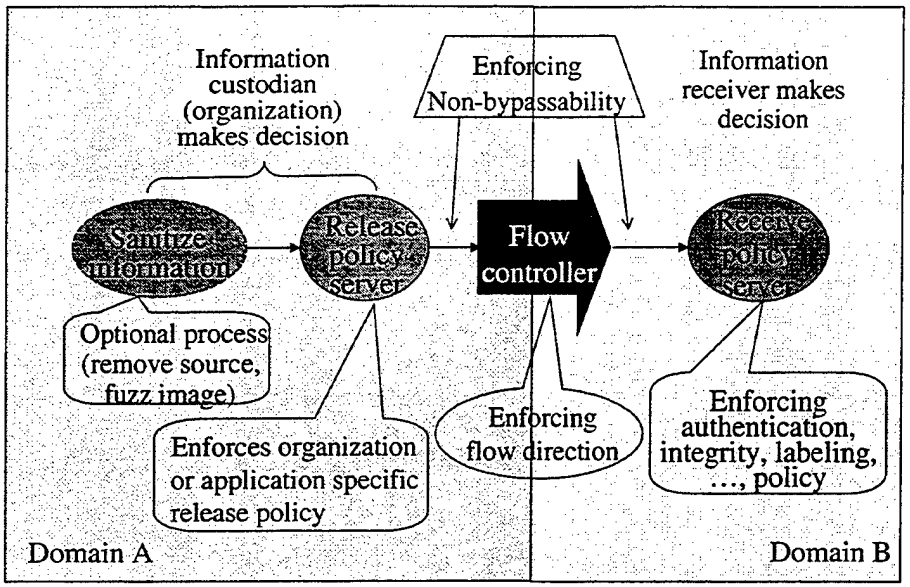


Figure 3: Information Release and Receive Policies in Conjunction with a Flow Controller

In general, when COTS software passes data to a flow controller a *wrapper* translates the protocol of COTS software to that of the flow controller because COTS software and flow controllers communicate with other software through their own protocols. Hence, a wrapper can be considered a protocol translator. The detailed description of the architecture and the MLS services are in [7].

Workflow Interoperability

As we mentioned earlier, our strategy for implementing an MLS workflow is through combining single-level workflows on an MLS distributed architecture. Workflows in different domains may be heterogeneous and autonomous. Hence workflow interoperability is an important requirement for the approach that we have taken to implement an MLS workflow. Two important aspects of workflow interoperability are:

- Interoperability protocol among independent WMFS.
- The ability to model interoperability in a workflow process definition tool (i.e., workflow designer).

A standard body such as Object Management Group (OMG) can handle the first aspect (e.g., jFlow [4]). However, the second aspect should be handled by each WFMS.

OMG's jFlow introduces two models of interoperability. They are nested sub-process and chained processes as shown in Figure 4-a and 4-b respectively.

In nested sub-process workflow structures, a task in workflow A may invoke workflow B as the performer of a task and then wait for it to finish. Hence, the task in workflow A is a requester, and the task that is realized by the sub-processes can serve as the synchronization point [11] for interaction of the two workflows.

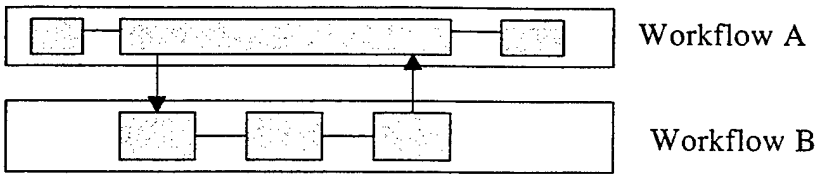


Figure 4-a: Nested Sub-Process

In chained workflow structures, one task may invoke another, then carry on with its own business logic. The workflows terminate independently of each other; in this case, the task registered with the sub-process would be another entity that is interested in the results of the sub-process.

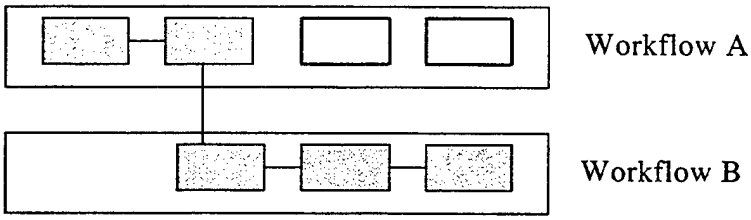


Figure 4-b: Chained Processes

The above two models provide powerful mechanisms for interoperability. However, we would like to extend these models to support an additional interoperability model, *cooperative processes*. Consider two independent autonomous workflows that need to cooperate. Let us assume that there are two cooperating organizations. Organization A is in charge of workflow A and Organization B is in charge of workflow B. Tasks in workflow A and workflow B can communicate and synchronize with each other as shown in Figure 5. In this example, two workflows may have independent starting and ending points.

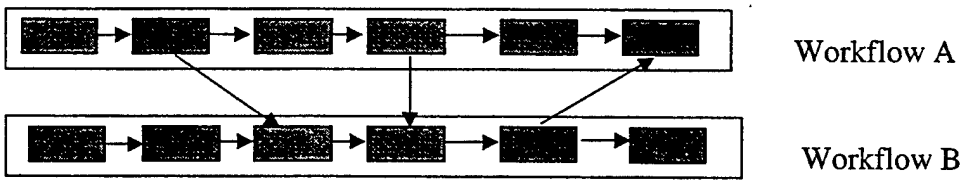


Figure 5: Cooperative Processes

There is another situation that we want to support in the context of cooperative processes. In general, the designer of workflow A does not need to know the structure of workflow B and vice versa. In conjunction with MLS principles, the designer of a workflow may not be allowed to know the detailed workflow structure of a higher level workflow. For example, in Figure 1, the designer of the workflow, whose classification domain is M, may not be allowed to know the workflow structure that contains the TSA_H task.

However, there is a minimal set of information that is required for communication and synchronization among tasks in cooperating autonomous workflows. This includes:

- Where and how to send/receive requests (i.e., the location and invocation method of tasks) and
- How and where to receive replies (i.e., expected output and the return location).

Therefore, the above specification has to appear in the workflow design so that the proper runtime code can be generated. Hence, we need a primitive that represents this situation in the design tool. This is the reason that we have introduced foreign tasks in the workflow model (section 4).

4. An MLS Workflow Model

Our strategy for implementing an MLS workflow is to combine single-level workflows on an MLS distributed architecture. We have chosen the METEOR WFMS [2, 3, 8, 10] as our single-level WFMS because it is a CORBA compliant, recoverable, and distributed WFMS (i.e., ORBWork [10] is a specific version of METEOR). METEOR also supports legacy tasks. It is an important feature for DoD because DoD has legacy applications that are costly to replace all at once. Hence, METEOR is a good starting point for extending capabilities to support MLS workflow.

To accommodate MLS workflow, the METEOR model has been modified. We summarize only the small subset of the new MLS METEOR model necessary for understanding this paper. A detailed description of the METEOR model can be found in [11].

In the METEOR model, a *task* represents an abstraction of an activity. A task can be regarded as a unit of work which is performed by a variety of processing entities, depending on the nature of the task. A task can be performed by (*realized by*) a human or by a computerized activity that executes a computer program, a database transaction, or possibly a network (workflow or subworkflow) of interconnected tasks. Hence, a task provides one level of abstraction (view) and its realization provides a lower level of abstraction (view). This also directly maps to the nested sub-process concept of jFlow. Since the realization of a task may contain many tasks at different levels of abstraction, a task is a recursive reference in the METEOR model.

In this paper, we categorize tasks into two types:

- *Foreign task:* A task whose realization (i.e., strategy for implementation) is unknown to the workflow designer. It represents a task that is a part of cooperating independent autonomous workflow. It is required for a designer to declare a foreign task explicitly to provide a hint to the METEOR runtime code generator. A foreign task should have a minimal information set that we specified in section 3 (e.g., invocation, output, where to send the request).
- *Native task:* A task for which the realization is known or the realization will be provided before the runtime code generation (i.e., all other tasks except the foreign tasks).

A *network task* represents the core of the workflow activity specification. Since a network task is one of the realizations of a task, it is always associated with a task called its *parent task*. A single network of tasks defines a relationship among workflow tasks, transferred data, exception handling, and other relevant information. It is a collection of either foreign or native tasks and transitions from one task to another. Figure 6 shows a simplified version of two levels of abstractions (views) where Task2 is the parent task of the projected workflow W_i which contains tasks 4, 5, 6, and 7, and transition t_j represents a transition from Task1 to Task2. In Figure 6, Task1, Task2, and Task3 may belong to different classification domains. Hence, the MLS METEOR model can be thought of as follows: along the *xy*-plane, there are tasks in different domains and along the *z*-axis, there are different levels of abstraction.

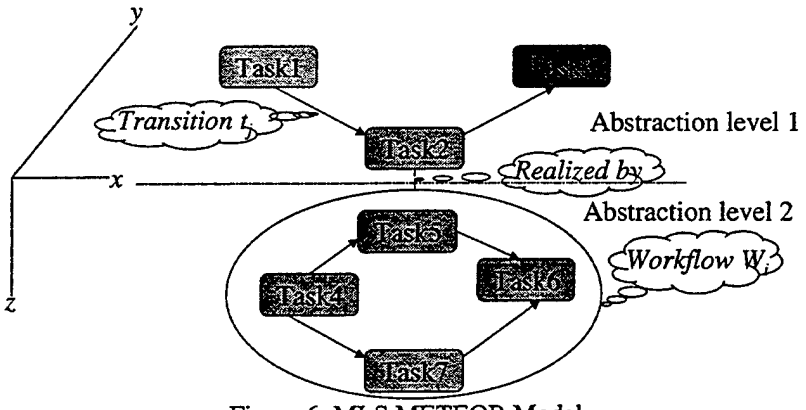


Figure 6: MLS METEOR Model

A task may play the role of a source task or a destination task (e.g., Task1 is the source task and Task2 is the destination task of the transition t_j in Figure 6) for a number of transitions. All of the transitions for which a task is the destination task are called the *input transitions* for that task (e.g., transition t_j is an input transition for Task2). Likewise, all the transitions for which a task is the source task are called its *output transitions* (e.g., transition t_j is an output transition of Task1). A transition may have an associated Boolean condition called its *guard*. A transition may be activated only if its guard is true. When there is a transition from task T_i to task T_j, where T_i and T_j are in different classification domains, we call this an MLS *transition* from T_i to T_j.

An *external transition* is a special type of a transition in which the two participating tasks (source and destination) are not in the same workflow (i.e., transition to and from a foreign task). An implied external transition may lead to a start task of another workflow. Similarly, an implied transition leads from the final task and is used to notify the external entity that the network has terminated. Note that an MLS transition is turned into an external transition when an MLS design is divided into multiple single-level workflows for runtime.

External transitions are also used to specify synchronization points with some external events. Typically, external transitions may be used to specify communication and synchronization between two independent workflows. Here, an external transition leading into a task in the workflow is assumed to have an implied source task (outside the workflow). Similarly, an external transition leading out of a task in the workflow is considered to have an implied destination task (outside the workflow). External transition is a cornerstone of our strategy to support MLS workflow.

The classes (i.e., types of objects) that are associated with an input transition to a task are called the task's *input classes*, and those appearing on an output transition are called *output classes* of that task. A task's output class, which is not its input class, is *created* by the task. Specifically, an object instance of the specified class is created by the workflow runtime. A task's input class, which is not its output class, is *dropped (consumed)*. When input classes are unused by the task, they are transferred to the task's successor(s).

A group of input transitions is called an *AND-join* if all of the participating transitions must be activated for the task to be *enabled* for execution. An AND-join is called *enabled* if all of its transitions have been activated. All the input transitions of a task may be partitioned into a number of AND-joins. A group of input transitions is called an *OR-join* if the activation of one of the participating transitions enables the task.

A group of transitions is said to have a *common source* if they have the same source task and all lead either from:

- Its success state or
- Its fail state.

A group of common source transitions may form either:

- *AND-split*: Each of the transitions in the group has the condition set to `true`. This means that all of the transitions in the group are activated once the task is completed.
- *OR-split* (selection): An ordered list of transitions where all but the last transition may have arbitrary conditions (i.e., the last transition on the list has the condition set to `true`). The first transition whose condition is satisfied will be activated.
- *Loop*: A special case of an OR-split, where the list is composed of exactly two transitions: loopback and continue. Loopback implies branch taken and continue implies branch not taken (i.e., fall through).

All tasks that we define in this paper are single-level tasks. What we mean by single-level is that the task receives input from one classification domain and produces output at the same classification domain. There are four special tasks: *begin, success, failure,* and *synchronization*. The synchronization tasks represent

external transitions to and from other workflows. In general, workflow designers do not manipulate synchronization nodes directly. They are automatically generated by the system based on the specification of foreign tasks and input and output transitions to and from the foreign tasks.

An MLS *workflow* is a network of interconnected single-level (foreign or native) tasks from more than one classification domain. Note that we call a task single-level from one particular level of abstraction (view). Since a single-level task may be realized by an MLS workflow at a lower level of abstraction, it may have side-effects on different classification domains at lower abstraction levels. Hence, our distinction between single-level and multilevel is purely from the perspective of a specific abstraction level.

Let $CL(T_i)$ represent the classification domain of task T_i. An MLS workflow that is the realization of task T_i where $CL(T_i) = S_a$ must obey the following constraints:

- The *begin, success,* and *fail* nodes of the MLS workflow must be $CL(begin) = CL(success) = CL(failure) = S_a$.
- It may have tasks in other classification domains; however, if the $CL(T_j) = S_b$ where S_a does not dominate S_b, then T_j must be a foreign task. In other words, only tasks in S_c where $S_a \geq S_c$ may have realizations.

For example, the workflow in Figure 1 is designed at domain M; thus special nodes are located in domain M. If the workflow designer creates an MLS workflow from the highest classification domain with a complete view of the workflow being designed, then the complete MLS workflow with realization of all its tasks can be specified. However, if the workflow designer creates an MLS workflow that requires input from (output to) higher classification domains, then he may only know the interfaces to the tasks at the higher levels but not the detailed workflow process at those levels. In such cases, foreign tasks can be used to define communication and synchronization with a task at higher classification domains.

5. MLS Dependence and MLS Workflow Decomposition

As we mentioned briefly in section 2, an MLS workflow design tool allows MLS workflow designers to:

- Divide a design area into multiple domains,
- Drop tasks in different domains, and
- Specify data and control flow among them.

Once the design of an MLS workflow is completed, the MLS workflow has to be divided into multiple single-level workflows to be executed on the underlying MLS distributed architecture that was described in section 3.

MLS Dependence

An MLS workflow design tool should allow the same kind of intertask dependence as a single-level workflow design tool (e.g., guards, input and output classes). However, some dependence in an MLS workflow may be specified across classification boundaries (i.e., MLS dependence). In other words, workflow state information and some values may have to move across classification boundaries during workflow execution. Hence, it is important to understand what the vulnerability is, whether it is easily exploitable and how to reduce it.

In our MLS WFMS, all information that has to move across classification domains must go though information release and receive policy servers and a high-assurance flow controller (e.g., Pump or downgrader [9,16]). We divide a transition between two tasks in different classification domains into a series of transitions. For example, if there is a transition from a task in domain 1 (T_{D1}) to a task in domain 2 (T_{D2}) as in Figure 7, then

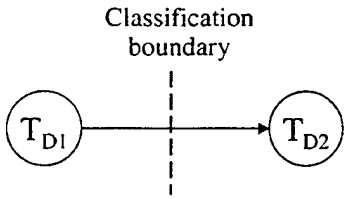


Figure 7: MLS Transition Across a Classification Boundary

this transition will be divided into transitions from T_{D1} to P_{D1}, P_{D1} to P_{D2}, and P_{D2} to T_{D2} as shown in Figure 8. Note that there is the flow controller between P_{D1} and P_{D2} where P_{D1} and P_{D2} are proxies that combine flow controller wrappers and policy servers (i.e., P_{D1} combines a flow controller wrapper and information release policy server, and P_{D2} combines a flow controller wrapper and information receive policy server).

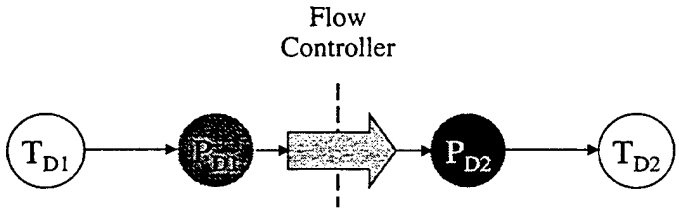


Figure 8: Indirect Transition Through a Flow Controller and Policy Servers

Note that domain policies may not allow combining flow controller wrappers and information policy servers. In that case, we have to break down transitions further. Also note that since T_{D1} and T_{D2} belongs to two different workflows, the P_{Di} serve as synchronization points that we discusses in section 4.

Decomposition of an MLS Workflow

Using the method that we described above, an MLS workflow will be separated into multiple single-level workflows. The single-level workflows neither communicate directly nor recognize single-level workflows from other classification domains. The number of single-level workflows that will be generated is equal to the number of classification domains in the MLS workflow (except the domains that contain only foreign tasks). When the breakdown is performed, direct transition (see Figure 7) becomes indirect transition as in Figure 8, and security depends on the underlying MLS architecture. Before we show an example of the division from an MLS workflow to multiple single-level workflows, we will formalize the MLS workflow model and the decomposition of an MLS workflow.

Formalism

An abstraction represents how we view a workflow. At the highest level of abstraction a workflow should be just one network task. However, as we traverse down the hierarchy of abstractions, workflows become more and more concrete with both network and simple tasks making up connected components via the flows. At the bottom of this hierarchy of abstractions, there should be only simple tasks.

There exists a partially ordered set of classification domains $\{D_i,\leq\}$ and a function $CL:\{Tasks\}\cup\{W\} \rightarrow \{D_i\}$.

There exists a set *Tasks*. (To distinguish the elements from the set itself, we capitalize the set name.) The set *Tasks* can be distinguished into the disjoint union of *Network Tasks* and *Simple Tasks*. The set of *Simple Tasks* can be further decomposed into TT ⊔ NT ⊔ HT. A simple task can be thought of as the

smallest unit of work from the workflow's point of view because it is only associated with only one level of abstraction. TT represents transactional tasks, NT are non-transactional tasks, and HT are human tasks. One can think of a network task as a convenient way to group tasks that are associated with multiple levels of abstractions.

There also exists a set *Flows*. A flow can be thought of as the order of program execution (control and data flows). We are interested in tasks and flows that can be formed into directed graphs with tasks being the vertices and flows being the edges.

A **workflow** W is a directed graph made up of tasks and flows. {W} is the set of workflows. The goal of this section is to formalize the ideas about what workflows represent in the rest of this paper, thus ensuring rigor. In this section the concept of "MLS" is implicit in a workflow or tasks. We express this by the following definition.

There exists a set of *Abstractions* $\{z_1, z_2, \ldots, z_n\}$, the abstraction levels, with an ordering $\leq$, such that $z_1 < z_2 < \cdots < z_n$.

A workflow is always associated with an abstraction level z_j. The notation W^j signifies this association. The tasks of W^j are $\{N^j_1, N^j_2, \ldots, N^j_{\alpha 1}, S^j_1, S^j_2, \ldots, S^j_{\beta 1}\}$ where N^j_i is a network task of W^j and S^j_k is a simple task of W^j. For $j = n$, W^n consists of a single network task N^n_1 and For $j = 1$, W^1 consists only of simple tasks.

For each abstraction z_j, $j > 1$ there is a projection function π_{j-1} which takes an N^j_i of some W^j and gives a workflow W^{j-1} (associated with z_{j-1}). Furthermore we have the following:

Non-Increasing Security Projection Condition: $CL(\pi_{j-1}(N^j_i)) \leq CL(N^j_i)$, and this comparison is well-defined.

In other words the projection of a network task of a workflow will never give a workflow of a higher classification domain. Note that there is no concept of projection of a simple task because there is no lower level of abstraction for the simple task.

Starting with a workflow W^j (associated with z_j) we may form the *family of* W^j, $\mathcal{F}(W^j)$. The family of W^j gives a view of the workflow at its present, and all lower, levels of abstraction. Before making this definition precise we first discuss some other concepts.

Given W^j we define $P_{j-1}(W^j)$ as
$P_{j-1}(W^j) = \{\pi_{j-1}(N^j_i), \forall N^j_i \in W^j\}$, we may then form $P_{j-2}(W^j)$ as
$P_{j-2}(W^j) = \{\pi_{j-2}(N^{j-1}_i), \forall N^{j-1}_i \in P_{j-1}(W^j)\}$; thus we may recursively define
$P_{j-3}(W^j), \ldots, P_1(W^j)$. Now we are ready to define the family of W^j.

$$\mathcal{F}(W^j) = \{W^j, P_{j-1}(W^j), \ldots, P_1(W^j)\}$$

Given any classification domain D_k we may form the filter function F_{D_k} as:
$F_{D_k}(\mathcal{F}(W^j)) = \{\forall N^j_i \in W^j \ni: CL(N^j_i) = D_k\} \cup \{\forall N^{j-1}_i \in P_{j-1}(W^j) \ni: CL(N^{j-1}_i) = D_k\} \cup \cdots$
$\cup \{\forall N^1_i \in P_1(W^j) \ni: CL(N^1_i) = D_k\}$

We may also view the filter as a vector $\mathbf{F}_{D_k}(\mathcal{F}(W^j))$ where the (j-h)-component is
$\{\forall N^{j-h}_i \in P_{j-h}(W^j) \ni: CL(N^{j-h}_i) = D_k\}$.

We have the following obvious result: $\cup_k F_{D_k}(\mathcal{F}(W^j)) = \mathcal{F}(W^j)$.

Similarly, we may also form F_{bD_k} and $\mathbf{F}_{bD_k}$ by using network tasks whose classification is equal to or below D_k, instead of simply equal to D_k. Note that it is possible that some components are empty. The

filter function provides a way to decompose multilevel workflow into single-level workflows, or a specified classification domain and below. Hence, the filter function provides a different view of multilevel workflows for users at different classification domains. For example, the view of a multilevel workflow at the unclassified level and the view of the same multilevel workflow at the secret level are different.

An Example Decomposition

Let us consider the simple MLS workflow shown in Figure 1 with classification domains L < M < H. Since this particular workflow is designed at domain M (special nodes, B, S, and F signify where the particular workflow was designed), all tasks in domain M and L, which is dominated by domain M, may be native tasks. Since the designer of the workflow in domain M may not know the detailed structure of the workflow in domain H, which dominates M, he can declare the TSA_H a foreign task. The specification of foreign task expresses interfaces (i.e., invocation methods and outputs) and where to send requests. The transitions, TSA to TSA_H, TSA_H to Ops1, Logistics to l-1, and l-2 to Ops1, and input and output classes that are associated with each transition define when and what kind of data will be passed to other workflows at different classification domains. After applying filter functions F_M and F_L, the runtime code generator generates the two workflows as shown in Figures 9-a and 9-b. The shaded proxies in Figure 9 represent the combination of policy server, flow controller wrapper and synchronization nodes that represent external transitions.

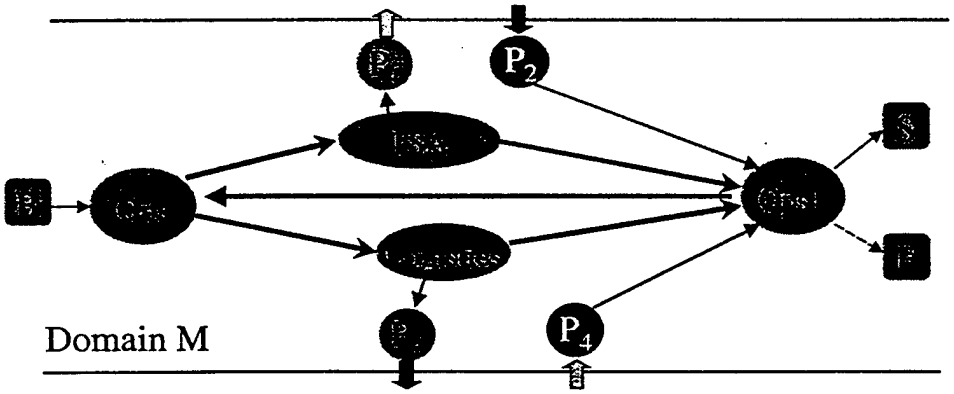


Figure 9-a: An Outcome of the M workflow after applying filter functions F_M

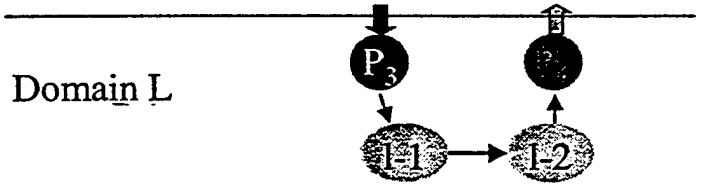


Figure 9-b: An Outcome of the L workflow after applying filter functions F_L

Note that workflows in domains M and L are two independent workflows after the filter function has been applied. The concept of cooperative processes and the synchronization node that we defined in sections 3 and 4 are useful for interoperability between these two independent workflows. Also note that applying the filter function to the original MLS workflow does not generate the workflow in domain H because there is only a foreign task in domain H.

6. Conclusion

Today's military is required to respond to constantly changing threats and cooperate with allies and different organizations. This dynamic environment and the military's dependence on IT systems require an MLS WFMS that allows

- Rapid specification and construction of mission specific IT systems,
- Maximum use of existing or commercial software/hardware, and
- Secure sharing and exchanging information among organizations in different classification domains.

In this paper, we presented requirements for an MLS workflow and tools needed to support an MLS WFMS. An MLS workflow designer should be able to specify different classification domains and tasks in those domains. He also should be able to specify control and data flows among tasks in different classification domains. To accommodate this need, we introduced an MLS workflow model and an MLS workflow design tool.

MLS workflow may be realized in many different ways. However, composing an MLS workflow from multiple single-level workflows is the only practical way to construct a high-assurance MLS WFMS today. Hence, we presented a strategy for implementing an MLS workflow by composing multiple single-level workflows on a multiple single-level architecture. When independent multiple single-level workflows work together to achieve a higher mission, workflow interoperability is a vital element. We introduced an extended workflow interoperability model for that purpose. We then presented a method for decomposing an MLS workflow design into multiple single-level workflows for runtime.

References

1. M. Anderson, C. North, J. Griffin, R. Milner, J. Yesberg, and K. Yiu, "Starlight: Interactive link," 12th Annual Computer Security Applications Conference, San Diego, CA, 1996.

2. N. Krishnakumar and A. Sheth, "Managing Heterogeneous Multi-System Tasks to Support Enterprise-Wide Operations," *Distributed and Parallel Database Journal*, 3 (2), April 1995.

3. METEOR project home page, http://lsdis.cs.uga.edu/proj/meteor/meteor.html.

4. OMG jFlow submission, ftp:///fpt.omg.org/pub/bom/98-06-07.pdf.

5. M. Kang , I. Moskowitz, and D. Lee, "A Network Pump," *IEEE Transactions on Software Engineering*, Vol. 22, No. 5, pp. 329 - 338, 1996.
6. M. H. Kang, J. N. Froscher, and I. S. Moskowitz, "An Architecture for Multilevel Secure Interoperability," 13th Annual Computer Security Applications Conference, IEEE Computer Society, San Diego, CA, 1997

7. M. H. Kang, J. Froscher, and B. Eppinger, "Toward an Infrastructure for MLS Distributed Computing," 14th Annual Computer Security Applications Conference, Scottsdale, AZ, 1998.

8. J. Miller, D. Palaniswani, A. Sheth, K. Kochut, H. Singh, "WebWork: METEOR's Web-based Workflow Management System," *Journal of Intelligent Information Systems,* Vol 10 (2), March/April 1998.

9. L.W. Chang and I.S. Moskowitz "Bayesian Methods Applied to the Database Inference Problem," 12th IFIP WG 11.3 Working Conference on Database Security, Chalkidiki, Greece, July 15-17, 1998.

10. K. Kochut, A. Sheth, and J. Miller "ORBWork: A CORBA-Based Fully Distributed, Scalable and Dynamic Workflow Enactment Service for METEOR," UGA-CS-TR-98-006, Technical Report, Department of Computer Science, University of Georgia, 1998.

11. M. Reichert M. and P. Dadam, "ADEPT$_{flex}$ - Supporting Dynamic Changes of Workflows Without Losing Control," *Journal of Intelligent Information Systems,* Vol 10 (2), March/April 1998.

12. Security Architecture for the AITS Reference Architecture, http://web-ext2.darpa.mil/iso/ia/Arch/SecArch1/SecArch1.htm

13. V. Atluri, W-K. Huang and E. Bertino, ``An Execution Model for Multilevel Secure Workflows" 11th IFIP Working Conference on Database Security, August 1997.

14. V. Atluri, W-K. Huang and E. Bertino, ``A Semantic Based Execution Model for Multilevel Secure Workflows," *Journal of Computer Security*, To appear.

15. R. Thomas & R. Sandhu "Task-based Authorization Controls (TBAC): A Family of Models for Active and Enterprise-oriented Authorization Management" 11th IFIP Working Conference on Database Security, August 1997.

16. I. S. Moskowitz and L. Chang, The Rational Downgrader, Proc. PADD'99, London, UK, April 1999.

Data Mining/Data Warehousing Session

Tuesday, July 27, 1999

Chair: Bhavani Thuraisingham
The MITRE Corporation

Impact of Decision-Region Based Classification Mining Algorithms on Database Security

Tom Johnsten
tdj5315@usl.edu
Computer Science Department
University of Southwestern Louisiana
Lafayette, LA 70504, USA

Vijay V. Raghavan
raghavan@cacs.usl.edu
Center for Advanced Computer Studies
University of Southwestern Louisiana
Lafayette, LA 70504, USA

Abstract

One of the challenges facing the computer science community is the development of techniques and tools to discover new and useful information from large collections of data. There are a number of basic issues associated with this challenge and many are still unresolved. This situation has led to the emergence of a new area of study called "Knowledge Discovery in Databases" (KDD).

The recent efforts of KDD researchers have focused primarily on issues surrounding the individual steps of the discovery process. Those issues that are not directly related to the discovery process have received much less attention. One such issue is the impact of this new technology on database security. In this paper, we investigate issues pertaining to the assessment of the impact of classification mining on database security. In particular, the security threat presented by a category of classification mining algorithms that we refer to as decision-region based is analyzed. Providing safeguards against this threat requires, in part, the development of new security policies. Our specific contributions are the proposal of a set of security policies for use in the context of decision-region based classification mining algorithms along with the specification and implementation of a security risk measure that allows for the realization of a subset of the proposed policies.

1 Introduction

Today many companies and organizations are collecting and storing vast amounts of data. They have discovered that large collections of data often contain valuable patterns or rules that can help them maintain their competitive edge. The development of techniques and tools to discover valuable patterns or rules presents a variety of technical challenges. This situation has led to the emergence of a new area of study called "Knowledge Discovery in Databases" (KDD) which has attracted the interest of researchers from a variety of fields, including statistics, pattern recognition, artificial intelligence, machine learning, and databases. Currently, the primary focus of KDD research is on issues surrounding the individual steps of the discovery process [1]. As a result, those issues that are not directly related to the discovery process have received little or no attention from the KDD research community. One issue, in particular, that has received only minimal attention is the impact of this new technology on database security.

Chris Clifton and Don Marks are two of a small number of researchers who have examined the potential impact of KDD technology on database security. In their paper, *Security and Privacy Implications of Data Mining,* Clifton and Marks outline several very general strategies designed to eliminate or reduce the security risk presented by this new technology [2]. Their strategies include allowing users access to only a subset of data, altering existing data or introducing additional (spurious) data. They contend that the application of such policies is most effective in the context of specific learning tasks. These tasks include classification, estimation, clustering, characterization and association [1,3,4]. Of special interest to the current work are the classification mining algorithms, which have the potential to disclose sensitive information whenever a database contains both "sensitive" and "non-sensitive" data [2]. Specifically, our current work has focused upon the need to develop security policies designed to minimize or eliminate the threat presented by classification mining in the context of the relational data model.

A valid security policy with respect to classification mining is the protection of all the attribute values of a

tuple whenever a tuple includes at least one sensitive, or *protected*, data element. That is, the tuple is entirely eliminated from the user's view. In general, such a policy unnecessarily restricts a user's access to the data. An alternative policy, which is the one we propose, is to protect only data elements that need to be concealed in order to prevent the disclosure of sensitive information through classification mining. This type of policy has the obvious advantage of allowing maximum use of the data and at the same time protecting sensitive information. However, the implementation of such a policy requires an accurate assessment of the data in order to determine a protected data element's risk of disclosure and the need to conceal additional data elements.

A possible assessment strategy in this case is to assess a protected data element's risk of disclosure in the context of a specific classification algorithm. Then, based on the results for several selected methods, a decision can be made with regards to a protected data element's risk of disclosure. An alternative assessment strategy is to make a generic assessment of a protected data element's risk of disclosure that is independent of a specific classification method. This strategy has a number of potential advantages over the former. These include:

- Producing security policies that are applicable to a general set of classification methods.
- Providing insight on how to modify the protection level of a protected data element.
- Reducing the time complexity of the process of assessing the risk of disclosure.

Unfortunately, a completely generic assessment that is independent of a specific classification method is in all likelihood an impossibility as a result of variations among classification mining algorithms. However, such an assessment becomes feasible when the scope of the evaluation is limited to a specific group of classification algorithms and/or certain restrictions are placed on the domain of the given attributes. Of course, the realization of this condition requires the partitioning of classification algorithms into groups that have uniform assessment properties of a protected data element. We have currently identified one such group of algorithms, which we will refer to as decision-region based.

The primary focus of this paper is on the assessment of a protected data element's risk of disclosure with respect to the decision-region based classification algorithms. To that end, the rest of this paper is organized as follows. Section two presents a general overview of classification mining along with an example to illustrate the security threat presented by classification mining algorithms. Section three proposes a set of security policies that when implemented have the potential to provide a high level of protection against classification mining and at the same time maximize access to the given data. Section four characterizes the decision-region based classification algorithms and describes the uniform assessment properties possessed by this group of algorithms. Section five proposes a security measure designed specifically for assessing a protected data element's risk of disclosure with respect to the decision-region based algorithms. Section six presents an outline of an evaluation algorithm, called Orthogonal Boundary (OB), which when executed results in the application of the proposed security measure against a relation instance. The application of the measure allows for the implementation of the security policies presented in section three. Section seven presents the results of experiments that were conducted in order to assess the validity of both the proposed security measure and a subset of the proposed security policies. Section eight briefly summaries the work presented in this paper and presents an outline of future research projects.

2 Classification Mining and Database Security

The goal of classification mining is to discover patterns that classify objects, or tuples in the context of the relational data model, into predefined classes [1,3]. This goal is achieved, in part, through the successful completion of three specific tasks. One of the required tasks is the selection of an attribute from the given relation. The selected attribute is typically referred to as the decision variable since its purpose is to partition tuples into disjoint sets or classes. Another required task is to generalize, if needed, the current values of the selected decision variable to form a set of named classes. Table 1 shows a generalized instance of a relation in which the values of the decision variable Mileage have been replaced by the class labels, *low*, *med* and *high*. The final required task is to partition the available data into two disjoint sets, a training set and a validation set [5]. The training set is analyzed by a classification mining algorithm to discover patterns that are relevant to the classification of objects into the predefined classes, while the validation set is used to judge the validity of the discovered patterns.

Fuel	Cyl	Power	Tran	Mileage
efi	4	high	manu	med
efi	6	high	manu	med
2-bbl	6	high	auto	low
efi	6	med	manu	med
efi	4	high	manu	high
2-bbl	4	med	manu	high
efi	6	high	auto	low
efi	6	med	manu	low
efi	4	med	auto	med
2-bbl	4	high	manu	high
efi	4	med	manu	med
efi	4	high	auto	high
2-bbl	4	low	manu	high
efi	6	high	auto	med

Table 1: Relation Instance.

Obviously, the generalizability (or predicatability) of the results are only as good as the extent of agreement of patterns or relationships between the training and validation sets as judged by the validation process.

A well known category of classification algorithms is the decision tree classifiers [4,5,6]. These algorithms are characterized by the formulation of a tree structure in which the interior nodes of the tree represent tests of a single attribute and the exterior nodes represent objects of a single class. A test on an attribute results in the partitioning of objects into mutually exclusive sets or outcomes. A simple decision tree generated from the data in Table 1 is shown in Figure 1. This tree was generated through the application of the C4.5 software package [6].

Of course, the purpose of a decision tree is to assign a correct class label to previously unseen objects. A class label is assigned to an object by starting at the root node of the tree and advancing downward through the tree until an exterior node is reached. For example, the assignment of a class label to the tuple, (Cyl = *4*; Fuel = *efi*; Power = *low*), results in the traversal of the path illustrated in Figure 1 as a dash line, and is assigned the class label *med*. It is common to refer to the logical conjunction of attribute name value pairs, corresponding to the path traversed, that defines the membership condition of the corresponding class as its *description*. In some cases, it may be desirable to transform a decision tree into a set of production rules. The production rules listed in Table 2 were obtained from the decision tree in Figure 1 and were also generated using the C4.5 software package. The percentage value associated with each rule is C4.5's predicted accuracy of the rule on classifying instances that satisfy the rule's left-hand side.

We now illustrate how the disclosure of sensitive information may occur through the execution of a classification mining algorithm. Our example is based on

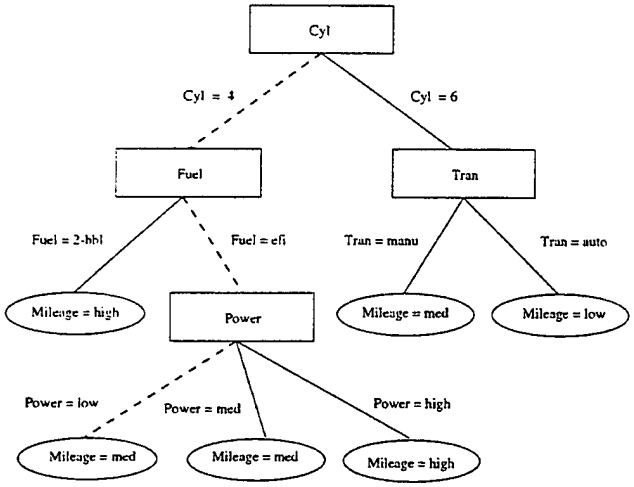


Figure 1: Decision Tree Generated From Table 1.

the data shown in Table 3. This table is an extension of Table 1 and includes the additional attributes Id and Prod, along with the three additional tuples T15, T16 and T17. Suppose that the car company that owns the data has implemented the following security policy: "junior engineers may not access the mileage class of pre-production cars". This policy might be the result of company officials attempting to reduce the chance that someone outside the company will learn the mileage class of a newly designed car. As a result, company officials have voluntarily released the data shown in Table 4 to all junior engineers. The only difference between Tables 3 and 4 is the presence of NULL values in the latter table. In this instance, a NULL value, referred to as a *protected data element*, protects the mileage class of a pre-production car. The Mileage attribute is referred to as the *protected attribute* since it contains the protected data elements; and, the attributes Id, Fuel, Cyl, Power, Prod, and Tran are referred to as *non-protected attributes* since they contain no protected data elements. Similarly, we refer to the tuples that contain a protected data element as *protected tuples*. In this case the protected tuples are T15, T16 and T17.

The security risk presented in this example is the extent to which the voluntarily released data facilitates the disclosure of a protected mileage value. The disclosure of that information can be achieved through the process of solving a classification problem. In other words, a junior engineer may be able to correctly infer a protected mileage value through the application of a classification mining algorithm to instances with a known mileage value. For example, access to the rule

Rule-1: IF (Cyl = 6) ∧ (Tran = auto) THEN (Mileage = low) [31.4%]
Rule-2: IF (Fuel = 2-bbl) ∧ (Cyl = 4) THEN (Mileage = high) [63.0%]
Rule-3: IF (Cyl = 4) ∧ (Power = high) THEN (Mileage = high) [45.3%]
Rule-4: IF (Fuel = efi) THEN (Mileage = med) [44.1%]

Table 2: Production Rules Obtained From Decision Tree in Figure 1.

set in Table 2 would allow a junior engineer to infer with a relatively high degree of confidence the protected data element of the protected tuple T15 since this tuple satisfies the antecedent of Rule-2 and the predicted accuracy of Rule-2 is higher than that of a simple naïve prediction that always predicts a *med* mileage value (since Pr(Mileage = *med*) = $\frac{6}{14}$). In contrast, the risk of disclosure of the protected data element in tuple T17 is relatively low with respect to the given rule set since it is assigned an incorrect class label by Rule-2.

This example motivates the need for security policies to minimize security violation through classification mining.

3 Inference Based Security Policies

It is possible to view the security threat presented by a classification mining algorithm in terms of the expected occurrence of an unauthorized inference [2]. Figure 2 shows a block diagram of an inference system defined in terms of a classification algorithm. The inputs into the system are a set of tuples having a defined security classification at or below some level L and a protected tuple that contains a protected data element with a defined security classification level at some level $\hat{L}$, where $\hat{L} > L$. The output of the system, referred to as a class-accuracy set, is a set of ordered pairs (d_i, a_i), where d_i is the i^{th} attribute value (class label) in the domain of the protected attribute, and a_i is the predicted accuracy, according to the classification mining algorithm, of assigning to the protected tuple the class label d_i. Suppose, for example, that d_1, d_2, d_3, and d_4 are the attribute values of a protected attribute. In this case, if a classification inference system produces the class-accuracy set, $\{(d_1, .5), (d_2, .2), (d_3, .8), (d_4, .1)\}$, then the protected tuple is assigned the class label d_1, d_2, d_3 and d_4 with predicted accuracy .5, .2, .8 and .1, respectively. In order to simulate

Id	Fuel	Cyl	Power	Prod.	Tran	Mileage
T1	efi	4	high	n	manu	med
T2	efi	6	high	n	manu	med
T3	2-bbl	6	high	n	auto	low
T4	efi	6	med	n	manu	med
T5	efi	4	high	n	manu	high
T6	2-bbl	4	med	n	manu	high
T7	efi	6	high	n	auto	low
T8	efi	6	med	n	manu	low
T9	efi	4	med	n	auto	med
T10	2-bbl	4	high	n	manu	high
T11	efi	4	med	n	manu	med
T12	efi	4	high	n	auto	high
T13	2-bbl	4	low	n	manu	high
T14	efi	6	high	n	auto	med
T15	2-bbl	4	high	y	auto	high
T16	efi	6	med	y	auto	low
T17	2-bbl	4	low	y	auto	med

Table 3: Data Maintained by Car Company.

the output of an inference system, the predicted accuracy values, a_i, are obtained through the application of a predicted accuracy measure and in general their sum need not equal one.

We have identified two general criteria for assessing the output of a classification inference system. The first criterion is based on a chosen threshold value. This criterion involves security policies that require predicted accuracy values to be below a specified threshold. The other criterion is to assess the output of the system based on a ranking of predicted class-accuracy values. This particular criterion involves security policies that require the ranked position of the protected data element to lie within a specified range. These two criteria are referred to as threshold and rank criteria, respectively.

The threshold and rank criteria have led to the development of four inference-based security policies. Two of the four policies, *maximum threshold* and *maximum range*, are defined independently of the predicted accuracy value of the protected data element. Specifically, an instance of a maximum threshold policy is *satisfied* for some threshold value, ε, and for some class-accuracy set, $\{(d_1, a_1), (d_2, a_2), ..., (d_n, a_n)\}$, if all a_i $(1 \leq i \leq n)$ are less than ε . In this paper, "ε" is assumed to represent either an organization defined constant, expression or function whose value depends, in part, on the desired level of protection. The other type of security policy, which is also defined independently of a protected data element, called maximum range, is *satisfied* for some threshold value, ε, and for some class-accuracy set, $\{(d_1, a_1), (d_2, a_2), ..., (d_n, a_n)\}$, if [MAX $(a_1, a_2, ..., a_n)$ - MIN $(a_1, a_2, ..., a_n)$] $< \varepsilon$. To illustrate the maximum threshold and maximum range policies consider again the class-accuracy set, $\{(d_1, .5), (d_2, .2), (d_3, .8), (d_4, .1)\}$. In

Id	Fuel	Cyl	Power	Prod	Tran	Mile
T1	efi	4	high	n	manu	med
T2	efi	6	high	n	manu	med
T3	2-bbl	6	high	n	auto	low
T4	efi	6	med	n	manu	med
T5	efi	4	high	n	manu	high
T6	2-bbl	4	med	n	manu	high
T7	efi	6	high	n	auto	low
T8	efi	6	med	n	manu	low
T9	efi	4	med	n	auto	med
T10	2-bbl	4	high	n	manu	high
T11	efi	4	med	n	manu	med
T12	efi	4	high	n	auto	high
T13	2-bbl	4	low	n	manu	high
T14	efi	6	high	n	auto	med
T15	2-bbl	4	high	y	auto	NULL
T16	efi	6	med	y	auto	NULL
T17	2-bbl	4	low	y	auto	NULL

Table 4: Voluntarily Released Data.

this particular case the maximum threshold and maximum range policies are satisfied (that is, no violation of policy) only if the specified ε-value is greater than .8 and .7, respectively.

The remaining two identified policies, *protected threshold* and *protected rank*, are both defined in terms of the predicted accuracy value of the protected data element. In particular, the protected threshold policy is *satisfied* for some threshold value, ε, and for some class-accuracy set, $\{(d_1, a_1), (d_2, a_2), ..., (d_n, a_n)\}$, if $a_i < \varepsilon$, where a_i is the predicted accuracy value associated with the protected data element. The protected rank policy is *satisfied* for some class-accuracy set, $\{(d_1, a_1), (d_2, a_2), ..., (d_n, a_n)\}$, if the ranked position of the protected data element is not within the range [L, U], where L and U are positive integers such that $1 \leq L \leq U$ and $L \leq U \leq |\{a_1, a_2, ..., a_n\}|$. We refer to the interval [L, U] as the *non-secure rank range*. We illustrate the protected threshold and protected rank policies using the previously given class-accuracy set, $\{(d_1, .5), (d_2, .2), (d_3, .8), (d_4, .1)\}$. In this case, assuming d_1 is the actual value of the protected data element, the protected threshold policy is satisfied, implying no violation of policy, only if the specified ε-value is greater than .5 and the protected rank policy is satisfied only if the specified non-secure rank range is [L= 1, U=1].

We previously defined a classification inference system in terms of a specific classification algorithm (Figure 2). This definition, however, can be extended to include a general class of classification mining algorithms. As mentioned in the introduction, this type of generalization requires a partitioning of classification algorithms into groups that have common properties that enable a generic assessment of the predicted accuracy values of a class accuracy set. In the next section we characterize one such group of algorithms.

4 Decision-Region Based Classification Algorithms

Decision-region based classification algorithms share a common set of properties that give rise to a uniform assessment of the predicted accuracy values of a class accuracy set. Before stating these properties, we first characterize the decision-region based class of algorithms. A classification algorithm, A, is a decision-region based algorithm if and only if the following two conditions are satisfied:

- *Condition-1*: It is possible to identify *a priori* a finite set of descriptions, D, in terms of the properties present in an object O such that the particular description d used by A to classify O is an element of D.

- *Condition-2*: The predicted accuracy of assigning an object O satisfying a description $d \in D$ to a class C is dependent on the distribution of class label C relative to all class labels associated with the objects that satisfy d.

The first condition leads to the property that the effective assessment of the security risk for decision-region based classification algorithms requires explicit or implicit determination of the predicted accuracy values of the class-accuracy set associated with each description $d \in D$. The second condition enables us to select a particular method of computing the predicted accuracy values of a class-accuracy set. In general, inference-based security policies may be applied at two levels. If it is known *a priori* that a particular description d will be selected relative to the protected tuple in a protected relation, then we can apply a policy just to that description. This case is referred to as the *description level* security policy. This form of evaluation is possible only if we wish to do the assessment for a particular classification algorithm. An alternative to this approach is referred to as *description space level* security policy. In this case, we must ensure that a chosen security policy is satisfied no matter which d is chosen by a class of classification algorithms.

Given the above definitions and *Condition-1*, we concluded that the specification of inference-based security for decision-region based classification algorithms should be carried out at the description space level. In the following section, we propose a measure for computing the predicted accuracy values of a class-

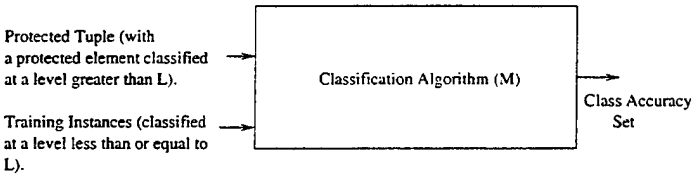


Figure 2: Classification Inference System.

accuracy set with respect to an arbitrary decision-region based classification algorithm.

5 Computing Predicted Accuracy Values

The proposed measure for computing the predicted accuracy values of a class-accuracy set in the context of a decision-region based algorithm is as follows. Let C be a class label in the domain of a protected attribute. Given a description d $\in$ D, the *predicated accuracy* a_i of assigning the protected tuple T the label C is the ratio of the number of tuples that are assigned label C and satisfy d to the number of tuples that satisfy d. The proposed measure is equivalent to the classification accuracy measure defined in [7].

We now illustrate the application of the measure using the description, (Fuel =*efi*) $\wedge$ (Cyl = *4*), applied against Table 1. In this instance there are zero tuples with a *low* gas mileage label that satisfy the description, three tuples with a *med* gas mileage label that satisfy the description, and two tuples with a *high* gas mileage label that satisfy the description. Thus, the predicted accuracy value for *low*, *med* and *high* is 0.0, 0.6 and 0.4, respectively. In the next section, we present the Orthogonal Boundary (OB) algorithm, which is, in part, designed to compute the class accuracy values with respect to an arbitrary description d $\in$ D that might be chosen by a specific subset of the decision-region based algorithms.

6 Orthogonal-Boundary (OB) Algorithm

The Orthogonal-Boundary (OB) algorithm has been designed for use with decision-region based classification algorithms that produce a specific type of class description. In particular, we require that each description, d $\in$ D, be a logical conjunction of attribute name value pairs that are sufficient, but may or may not be necessary. This type of description is produced by decision tree classifiers.

It follows that the set of such descriptions D, with respect to a protected tuple T, is the set of all logical conjunctions formed from one or more non-protected attribute name value pairs that appear in T. We refer to this set of descriptions as the description space, D*, of the protected tuple T. To illustrate the formulation of a description space, D*, consider the following protected tuple: (Fuel = *efi*; Cyl = *4*; Tran = *manu*; Mileage = *null*). In this case, D* is the set of logical expressions: {(Fuel = *efi*), (Fuel = *efi*) $\wedge$ (Cyl = *4*), (Fuel = *efi*) $\wedge$ (Tran = *manu*), (Cyl = *4*) $\wedge$ (Tran = *manu*), (Cyl = *4*), (Tran = *manu*), (Fuel = *efi*) $\wedge$ (Cyl = *4*) $\wedge$ (Tran = *manu*)}. It follows from *Condition-1* that the assignment of a class label to this tuple, by a decision-region based algorithm that produces a description space equivalent to D*, is necessarily a label that it associates with one of the seven listed descriptions.

Obviously, there is no way to identify *a priori* the description d $\in$ D* chosen by a classifier without making explicit assumptions about the operation of such an algorithm. Unfortunately, the number of descriptions belonging to a protected tuple's description space, D*, is exponential in terms of the number of non-protected attributes. There are, however, several conditions that can be taken advantage of to reduce the number of inspected descriptions (e.g. reduce the size of the search space).

One condition is the recognition of a special set of descriptions that we refer to as "zero" descriptions. The classes constructed from these descriptions contain no tuples with a class label corresponding to the protected data element. The recognition of a zero description implies that there is no need to inspect any description that is a specialization of the zero description since the resulting class will also contain zero instances of the protected data element. Suppose that the description (Tran = *manu*) is a zero description with respect to the protected tuple, (Fuel = *efi*; Cyl = *4*; Tran = *manu*; Mileage = *null*). In this situation, there is no need to inspect the descriptions: (Fuel = *efi*) $\wedge$ (Tran = *manu*), (Cyl = *4*) $\wedge$ (Tran = *manu*), and (Fuel = *efi*) $\wedge$ (Cyl = *4*)

Another condition that can also reduce the number of inspected descriptions is the transformation of a *non-secure description* into a *secure description.* A description is considered secure if its computed class-accuracy set satisfies the chosen security policy. Based on our measure of predicted accuracy and either a protected threshold or protected rank security policy, a

transformation of a non-secure description into a secure description requires a percentage reduction in the number of tuples satisfying the description with a class label equal to the protected data element. Obviously, such a reduction occurs when either the number of tuples satisfying the description with a class label equal to the protected data element is decreased, or the number of tuples satisfying the description with a class label that is not equal to the protected data element is increased. A possible transformation scheme, especially when the objective is to maximize the amount of accessible data without altering non-protected data values, is to "protect" additional values of the protected tuple so as to prevent the assignment of the tuple to the class defined by the non-secure description. This particular solution has the added benefit of reducing the required number of inspected descriptions. For example, if (Cyl = *4*) is a non-secure description with respect to the protected tuple, (Fuel = *efi*; Cyl = *4*; Tran = *manu*; Mileage = *null*), then by denying the tuple membership to the set defined by (Cyl = *4*) there is no need to inspect the descriptions: (Fuel = *efi*) $\wedge$ (Cyl = *4*), (Cyl = *4*) $\wedge$ (Tran = *manu*), (Fuel = *efi*) $\wedge$ (Cyl =*4*) $\wedge$ (Tran = *manu*).

Unfortunately, the protection of additional attribute values of a protected tuple T, in general, causes a decision-region based algorithm to violate *Condition-1* in the assignment of a class label to tuple T. Such a case occurs in the application of C4.5's "consult" interpreter which is designed to classify previously unseen tuples based on a constructed decision tree and to output a ranking of the possible class labels that correspond to the tuple. A feature of the "consult" interpreter is its ability, through the use of conditional probabilities, to assign a class label to a tuple that contains unknown, or protected, attribute values. It is this latter feature that potentially results in a violation of *Condition-1* since the assignment of a class label to a protected tuple T may not be based on a description d $\in$ D*. An alternative scheme to transforming a non-secure description into a secure description is to protect a subset of attribute values not belonging to the protected tuple. This solution requires the protection of attribute values such that a decrease occurs in the number of tuples satisfying the non-secure description with a class label equal to that of the protected data element. The advantage of the alternative scheme is that it ensures that the assignment of a class label to a protected tuple satisfies *Condition-1*; however, this scheme does not support maximum access to the data. The current implementation of the OB algorithm adheres to the first transformation scheme, the protection of additional attribute values of the protected tuple.

A third condition that can reduce the number of inspected descriptions is the establishment of an upper bound on the number of descriptions. By statically or dynamically protecting a subset of a protected tuple's non-protected attribute values, we can reduce the size of the tuple's description space. Of course, the disadvantage of such a strategy is that it does not guarantee maximum access to the data.

A conceptual version of the OB algorithm is shown in Figure 3. A k-description represents a description defined in terms of k attributes. Each iteration through the outer loop generates a set of descriptions that require inspection. The inner loop consists of a nested-if statement that evaluates the security status of a description. If a zero-description is faced then conceptually all specializations of that description are identified and placed onto the "zero-description" list. Otherwise, if a description is non-secure then the attributes comprising the description are identified and placed onto the "candidate-protect" list. The non-secure descriptions identified at level k are transformed into secure descriptions following the inspection of all k-descriptions. The current implementation of the OB algorithm requires human assistance in performing the transformations. Specifically, the algorithm displays a list of all non-secure descriptions at level k and prompts the user (the person responsible to set security policies) to select an appropriate set of the protected tuple's non-protected attribute values to conceal. The objective is to conceal a set of non-protected attribute values that deny the protected tuple's membership to the individual sets defined by the k-level non-secure descriptions and at the same time maximize the amount of accessible data. The result of executing the OB algorithm is the implicit or explicit inspection of a protected tuple's description space. The inspection process ensures that all descriptions belonging to the tuple's description space satisfy the user's specified description level security policy.

7 Experimental Investigation

In this section, experiments are conducted to validate a proposed approach to establish security policies based on the proposed class-accuracy measure and their results are reported. The objective of the experiments is to test the following hypothesis: there exist a protected threshold policy or a protected rank policy

```
Algorithm OB

Input:

      T  = protected tuple
      R  = relation


Initialization:

      zero_list = { }
      candidate_protect_list = { }
      protect_attr_list = { }
      k = 1
      A = { r is an element of k-description | (r has no attributes belonging to protect_attr_list)
                              &   (r is not an element of zero_list)}

Loop until (A is empty)

  Loop for each (r element of A)
    If (r == "zero" description) Then
      Append all specializations of r to zero_list
    Else
      If (r == "non-secure" description) Then
        Append all attributes of r to candidate_protect_list
    Endif
  Endloop
  If (candidate_protect_list is not empty) Then
     Transform "non-secure" description into "secure" description
     Update protected_attr_list
     Initialize candidate_protect_list = { }
  Endif
  k = k + 1
  A = {r is an element of k-description | (r has no attributes belonging to protect_attr_list)
                              &  (r is not an element of zero_list)}

Endloop
```

Figure 3: Orthogonal Boundary (OB) Algorithm.

applied at the description level that produces a protected rank policy at the description space level with a non-secure rank range of [L = 1, U = 1]. In other words, we wish to identify a description level protected threshold policy and/or a protected rank policy that, when applied to the individual descriptions of a protected tuple's description space, results in an appropriate description space level protected rank policy. In general, the determination as to whether a specific description space level policy has been successfully implemented must be based on an evaluation of the output from one or more decision-region based algorithms that have been applied to the protected tuple. In conducting the experiments, we restricted the evaluation of the description space level protected rank policy, [L = 1, U = 1], to the C4.5 decision tree classifier. The application of the classifier is described in the next section.

We anticipate that the implementation of the description space level protected rank policy, [L = 1, U = 1], will provide a high level of protection. This statement is based on the assumption that a user will assign the protected tuple, or more specifically the protected data element, the class label that is assigned the top rank by the chosen decision-region based algorithm. In addition, the non-secure rank range, [L = 1, U = 1], introduces a relatively high degree of uncertainty in a user's assignment of a class label to a protected tuple. This is because, even a user who has knowledge of the fact that the implemented description space level protected rank policy is [L = 1, U = 1] can only logically eliminate from consideration the class label that has been assigned the top rank by the decision-region based algorithm. Hence, a user is at best forced to make a random guess from n - 1 class labels; where n is the number of possible labels.

7.1 Experimental Parameters

The execution of the experiments required the construction of several *protected relation instances.* We define a protected relation instance as a relation consisting of at least one non-protected attribute and exactly one protected data element. A relation instance that contains n-protected data elements is viewed as n-instances of a protected relation. The protected relations used in this investigation were constructed through the insertion of a protected tuple into non-protected relation instances constructed through the execution of the Synthetic Classification Data Set (SCDS) program [8]. A total of four non-protected relation instances were constructed through the use of the program and each instance was produced with the parameter values shown in Table 5. The values of an irrelevant attribute have only a random relationship with the decision variable; and, a masked relevant attribute is an attribute whose values have a direct relationship with the decision variable, but the attribute itself is not included as part of the relation.

The protected tuples, unlike the non-protected relation instances, were manually generated from randomly selected attribute values. Specifically, six protected tuples were constructed with respect to Relation #1, six with respect to Relation #2, four with respect to Relation #3, and five with respect to Relation #4. Thus, a total of twenty-one protected relation instances were generated. Each protected tuple was evaluated against a set of protected threshold and protected rank policies applied at the description level. The implemented protected threshold policies included those defined at an ε-value of 0.9, 0.8, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2, and 0.1; and, the implemented protected rank policies included those defined at a non-secure rank range of [L = 1, U = 1], [L = 1, U = 2], [L = 1, U = 3] and [L = 1, U = 4]. These description level policies were applied to each protected tuple, T, through the execution of the OB algorithm. The result in each case was the *transformation* of a protected tuple, T, into a protected tuple T' such that each description, d, belonging to T''s description space D^* satisfied the stated policy as defined in section three. In general, the implementation of the individual description level policies resulted in a protected tuple, T', that contained multiple protected attribute values.

In order to assess the generality of the implemented description level policies, two distinct decision tree models were generated for each of the four non-protected relation instances. One set of models corresponded to the *gain attribute selection* criterion, while the other set of models corresponded to the *gain ratio attribute selection* criterion [6]. These two criteria are both supported by C4.5 and provide a decision rule for selecting the interior nodes of a decision tree. Each protected tuple T' was evaluated against instances of both models using C4.5's "consult" interpreter [6]. In conducting the experiments, an attribute value was specified as unknown if the interpreter requested a protected attribute value.

7.2 Experimental Results

The results of the experiments, with respect to nineteen of the twenty-one protected relation instances,

are shown in Tables 6 - 10. Tables 6 - 9 display the average, highest and lowest rank positions of the protected data elements across all nineteen protected relation instances. Specifically, Tables 6 and 7 display statistics about the rank positions produced by the "consult" interpreter when applied to the individual protected tuples that were generated by the application (description level) protected threshold policies using the OB algorithm. Tables 8 and 9 display the same statistics for the rank positions produced by the "consult" interpreter when applied to the individual protected tuples that were generated from the application (description level) protected rank policies using the OB algorithm. Table 10 displays the percentage of attribute values (among which all but one were non-protected), belonging to the individual protected tuples, that were mandated to be protected through the execution of the OB algorithm in order to satisfy the various (description level) protected threshold policies and the protected rank policies shown. It is interesting to note that the protected rank policies resulted in a relatively high percentage of protected attribute values as compared to the protected threshold policies.

The recorded rank positions in Table 6 state that a (description space level) protected rank policy that is satisfied by the non-secure rank range, [L = 1, U = 1], is realizable through the application of a (description level) protected threshold policy defined at either a threshold value (ε) of 0.4 or 0.5. In addition, the recorded rank positions indicate that a (description level) protected threshold policy defined at an ε-value other than 0.4 or 0.5 produces an *undefinable* (description space level) protected rank policy as a result of a maximum recorded rank position of one. The results recorded in Table 7 are almost identical to the results recorded in Table 6. The only significant difference is that a (description space level) protected rank policy that is satisfied by the non-secure rank range, [L = 1, U = 1], is realizable through the application of a (description level) protected threshold policy defined at a threshold value (ε) of 0.2 as well as 0.4 and 0.5.

The recorded rank positions in Table 8 state that a (description space level) protected rank policy that is satisfied by the non-secure rank range, [L = 1, U = 1], is realizable through the application of a (description level) protected rank policy defined with respect to either the non-secure rank range [L = 1, U = 2] or [L = 1, U = 3]. The remaining (description level) protected rank policies listed in Table 8 produced undefinable (description space level) protected rank policies. Surprisingly, the results recorded in Table 9 are significantly different from those in Table 8. In particular, the recorded rank positions in Table 9 state that a (description space level) protected rank policy that is satisfied by the non-secure rank range, [L = 1, U = 1], is realizable through the application of a (description level) protected rank policy defined with respect to either the non-secure rank range [L = 1, U = 1] or [L = 1, U = 3]. Again the remaining (description level) protected rank policies listed in Table 9 produced undefinable (description space level) protected rank policies.

7.3 Experimental Conclusions

If the assumption is made that a valid protected rank policy (description space level) is one defined with respect to a non-secure rank range, [L = 1, U = 1], then a valid protected rank policy is achievable based on the above results through the application of either a (description level) protected threshold policy defined at a threshold value (ε) of approximately 0.45 or a (description level) protected rank policy defined with respect to the non-secure rank range [L = 1, U = 3]. The implementation of the (description level) protected rank policy with a non-secure rank range, [L = 1, U = 3], resulted in a substantially higher percentage of protected attribute values having to be hidden than did the implementation of the (description level) protected threshold policy defined at an ε-value of 0.4 or 0.5. We suspect that when a (description level) protected threshold policy is defined in terms of either a relatively high ε-value (0.9, 0.8, 0.7, or 0.6) or a relatively high non-secure rank range ([L = 1, U = 1] or [L = 1, U = 2]), the percentage limit on the number of tuples with a class label equal to the protected data element is insufficient to ensure an adequate level of protection at the description space level. On the other hand, description level protected threshold policies defined in terms of a low ε-value (0.3, 0.2, or 0.1) or a low protected rank policy ([L = 1, U = 4]) over protect the protected data element. As a result, the assignment of a class label to a protected tuple is based entirely upon the dominant relationships, or patterns, that exist within the data, independent of the accessible attribute values of the protected tuple.

The two protected relation instances not represented in Tables 6 - 10 are exceptions to the notion of a valid (description space level) protected rank policy defined in terms of a (description level) protected threshold policy or a protected rank policy. Specifically, the rank position of the two tuples' protected data element as specified by the "consult" interpreter consistently occupied the top position across all implemented (description level) protected threshold and

	Relation #1	Relation #2	Relation #3	Relation #4
# Tuples	5000	5000	5000	5000
# Classes	5	5	5	5
# Relevant Attrs.	15	10	15	15
# Irrelevant Attrs.	3	0	2	0
# Masked Relevant Attrs.	1	2	0	3

Table 5: Parameter Values Used to Construct Non-Protected Relation Instances.

Threshold (ε)	Average Rank	Highest Rank	Lowest Rank
1.0	2.00	1	5
0.9	2.76	1	5
0.8	3.05	1	5
0.7	2.86	1	5
0.6	3.19	1	5
0.5	3.33	2	5
0.4	3.52	2	5
0.3	3.14	1	5
0.2	4.10	1	5
0.1	3.48	1	5

Table 6: Rank Positions With Respect to Protected Threshold Policies (Gain Ratio Criterion).

Threshold (ε)	Average Rank	Highest Rank	Lowest Rank
1.0	2.67	1	5
0.9	2.86	1	5
0.8	3.05	1	5
0.7	3.19	1	5
0.6	3.38	1	5
0.5	3.48	2	5
0.4	3.52	2	5
0.3	3.33	1	5
0.2	4.00	2	5
0.1	3.67	1	5

Table 7: Rank Positions With Respect to Protected Threshold Policies (Gain Criterion).

Non-Secure Rank Range	Average Rank	Highest Rank	Lowest Rank
L = 1, U = 1	3.53	1	5
L = 1, U = 2	3.79	2	5
L = 1, U = 3	3.84	2	5
L = 1, U = 4	4.32	1	5

Table 8: Rank Positions With Respect to Protected Rank Policies (Gain Ratio Criterion).

Non-Secure Rank Range	Average Rank	Highest Rank	Lowest Rank
L = 1, U = 1	3.68	2	5
L = 1, U = 2	3.74	1	5
L = 1, U = 3	4.16	2	5
L = 1, U = 4	4.16	1	5

Table 9: Rank Positions With Respect to Protected Rank Policies (Gain Criterion).

protected rank policies. We refer to such protected relation instances as *inherently non-secure*. In the case of such a relation instance the only logical course of action is to entirely eliminate the protected tuple from the user's view. Our preliminary work (not reported in this paper) in this area indicates that such relation instances are avoidable if the transformation of a non-secure description to a secure description is accomplished by protecting additional attribute values not belonging to the protected tuple (no violation of *Condition-1*); or, such relation instances may be identifiable through the application of an alternative predicated class-accuracy measure.

8 Conclusion and Future Work

The work presented in this paper concerned the accurate assessment of a protected data element's risk of disclosure with respect to the decision-region based classification mining algorithms. To that end, a new set of security policies were developed for use in this context along with the specification and implementation of a security risk measure that allows for the realization of a subset of the proposed policies. A set of experiments were also conducted in order to validate the overall quality of the proposed approach. Based on the results, we have concluded that a valid security policy is indeed achievable using the approach presented in this paper.

We have several research projects planned with respect to this new and challenging research area. Our immediate plans include, addressing the issue of inherently non-secure relation instances, improvement of the efficiency of the OB algorithm, mapping of continuously valued attributes on to the description space, D^*, of a protected tuple, and development of additional security measures for other groups of classification mining algorithms.

Description Level Policies	Protected Att.
Protected Threshold (ε) = 0.9	29.6
Protected Threshold (ε) = 0.8	30.4
Protected Threshold (ε) = 0.7	31.2
Protected Threshold (ε) = 0.6	33.6
Protected Threshold (ε) = 0.5	36.0
Protected Threshold (ε) = 0.4	43.0
Protected Threshold (ε) = 0.3	52.2
Protected Threshold (ε) = 0.2	66.4
Protected Threshold (ε) = 0.1	89.3
Non-Secure Rank Range [L = 1, U = 1]	54.9
Non-Secure Rank Range [L = 1, U = 2]	68.4
Non-Secure Rank Range [L = 1, U = 3]	76.3
Non-Secure Rank Range [L = 1, U = 4]	83.4

Table 10: Percentage of Protected Attributes Across All Protected Relation Instances.

References

[1] U. Fayyad, G. Piatetsky-Shapiro and P. Smyth, "From Data Mining to Knowledge Discovery: An Overview," in *Advances in Knowledge Discovery and Data Mining* (U. Fayyad, G. Piatetsky-Shapiro, P.Smyth and R. Uthurusamy, eds.), Cambridge, MA, pp. 1-34, 1996.

[2] C. Clifton and D. Marks, "Security and Privacy Implications of Data Mining," in *1996 SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery*, British Columbia, pp. 15-19, 1996.

[3] J. Deogun, V. Raghavan, A. Sarkar, and H. Sever, "Data Mining: Trends in Research and Development," in *Rough Sets and Data Mining: Analysis for Imprecise Data*, Boston, MA, pp. 9-45, 1996.

[4] M. Berry and G. Linoff, *Data Mining Techniques: For Marketing, Sales, and Customer Support*, New York, NY: Wiley & Sons, 1997.

[5] T. Mitchell, *Machine Learning*, New York, NY: McGraw-Hill, 1997.

[6] J. Quinlan, *C4.5: Programs For Machine Learning*, San Mateo, CA: Morgan Kaufmann, 1993.

[7] M. Holsheimer and A. Siebes, *Data Mining: The Search for Knowledge in Databases*, Report CS-R9406, Computer Science, CWI, Amsterdam, The Netherlands, 1994.

[8] "Synthetic Classification Data Sets," http://fas.sfu.ca/cs/people/GradStudents/melli/SCDS/intro.html.

Protecting Against Data Mining through Samples

Chris Clifton

The MITRE Corporation
202 Burlington Road
Bedford, MA 01730-1420 USA
clifton@mitre.org

Abstract

Data mining introduces new problems in database security. The basic problem of using non-sensitive data to infer sensitive data is made more difficult by the "probabilistic" inferences possible with data mining. This paper shows how lower bounds from pattern recognition theory can be used to determine sample sizes where data mining tools cannot obtain reliable results.

1 Introduction

The problem of inference has received considerable attention in the database security community. The basic problem is using non-sensitive, or "low", data to infer sensitive, or "high" facts. As an example, we may want to keep the presence of a particular type of equipment (e.g., "super-secret aircraft" (SSA)) at a particular location ("Secret Base", SB) secret. However, to support logistics we want to make information about supplies needed by *all* bases available. The inference problem occurs if parts that are only used for the SSA are ordered by SB – from this we can infer that there must be a SSA at SB.

Most of the work in preventing inference in multilevel secure databases has concentrated on preventing such "provable" facts [4]. Recent work has extended this to capturing data-level, rather than schema-level, functional dependencies [16]. However, data mining provides what could be viewed as *probabilistic* inferences. These are relationships that do not always hold true (are not a functional dependency), but hold true substantially more often than would be expected in "random" data. Preventing this type of inference detection is beyond the reach of existing methods.

As an example, what if there are no parts that are used *only* for the SSA? The functional dependency inference problem no longer exists. However, there may be some items that are used *more heavily* by the SSA than other aircraft (e.g., it uses a great quantity of fuel). Data mining could find such a relationship; for example bases X, Y, and SB use an unusual quantity of fuel *in relation to other supplies.* If we know that bases X and Y support the SSA, we can make a good guess that SB does as well.

Hinke and Delugach [8, 9] give a breakdown of inference into seven classes of problems. The first six rely on combining known rules (e.g., Part A is only used on an SSA) with non-sensitive data (Part A was suppled to base SB) to infer sensitive facts. Class 7 is the inference of a sensitive rule. It is noted that this "represents a considerably different target than the previous ones", and as a result has received considerably less attention in the database security community. However, one of the primary focuses of data mining technology is inferring rules, with the rise of data mining technology, this has become a recognizable problem.

What can we do about this, in particular when we don't know what the adversary will be looking for? We can ensure that *any* results will be suspect. If we can convince the adversary that any "guess" (inferred rule) gained from data mining is no more likely to be a good guess than a random guess, the adversary will not be able to trust any data mining results.

How do we do this? Inferences from data mining have some level of confidence and support. We will show how to ensure that either:

1. The rules "discovered" may be incorrect (the levels of confidence and support could be significantly off); or
2. It is likely that rules exist with higher confidence and support than those found.

We propose to accomplish this by ensuring that the data available to the adversary is only a sample of the data on which the adversary would like the rules to hold (note that in some cases this "sample" may be

an entire database, as long as the "inferences" we wish to protect against apply to a larger population than that reflected in the database). We show that we can draw a relationship between the sample size and the likelihood that the rules are correct.

We base this on the fact that inference rules can be used to classify. Vapnik[14] has shown error expectations in classification on samples. This is due to expectations of a random sample of a population having a different distribution with respect to any classification information than the population as a whole. If we can show that a "best possible" classifier is likely to be incorrect, we can work back to show that any rules (at best a "best possible" classifier) will also likely be incorrect.

We divide this into two variations:

1. We are concerned with protecting a particular "object" (e.g., any relationship where the target is a single value for a given variable). Note that we do not require predefining the value. We can reduce the problem of learning a binary classifier to this.

2. We are concerned with a particular class of rules (e.g., inference rules involving a single independent variable). This limits our space of possible classifiers.

As we will show; these two variations lead to different solutions.

1.1 What this work will lead to

The eventual goal of this work is to provide a tool usable by security administrators. This tool would enable the administrator to determine:

- The allowable size of a sample, given constraints on the quality of what an adversary would be allowed to learn; and

- The quality of what an adversary would be allowed to learn, given the size of a sample.

To make this a usable tool, we must be able to abstract away from technical details of learning/data mining algorithms. We view certain information as clearly reasonable for a security administrator to provide:

$|D|$ The number of items in the database (or in the "world" about which we are concerned; if we are concerned about the application of discovered knowledge to the real world, and the database is already only a sample of the real world).

n The number of items in the sample (unless this is the goal of our calculation).

Attributes The number and type (e.g., continuous, discrete with k possible values) of attributes for the entities in the data.

Note that this assumes a "single table" view of the data; this matches current data mining technology. For a complete database, independent analyses can be performed for each table (or collection of tables that describe a single type of entity).

We need other information that is more difficult to provide, part of the focus of this work is defining this information in a form reasonable for security administrators.

Mining type To obtain tight bounds on error given a sample, we need to know the descriptive power of the adversary's data mining technology. This can be assumed to be "worst case", however if the security administrator knows the *type* of knowledge that needs to be protected against (e.g., inference rules that can classify the data into two groups) we can obtain better bounds.

Expected error To give a recommendation on sample size, we need to know the expected error we plan to force an adversary to live with. This can be difficult; for example for an inference rule that does binary classification, a statement like "The rule must be wrong 25% of the time" would be unreasonable if 90% of the cases belonged to one class. Better descriptions might be "There is a 25% expectation that the actual best rule will be better than the best rule found on the sample", or "The actual confidence of any rule can be expected to have 25% error".

Note that the means for describing expected error will be dependent on mining type; one possibility would be to have a security administrator provide error for one type and have the system determine expected error for other types of mining.

This paper will only provide rudimentary examples of how such a security administration tool would appear. The focus is on establishing sample size/error relationships for a simple class of problems (binary classification), and giving an example of how this may be extended. There is clearly substantial room for further work in this area.

1.2 Applications and problem extensions

Providing a random sample of data has limited utility. A few examples of where this *has* utility are:

- Development: We can provide a sample of real data to a system developer to use in design/testing. This allows the developer to do

a better job of testing, without clearing them for access to the entire database. This requires clearing the data *items* and checking for functional-dependency type inferences, but these alone would not be sufficient. The techniques prevented here are necessary to ensure that such a "sanitized sample" doesn't contain *hidden* rules.

- Damage assessment: If a portion of the database *is* released, we can analyze potential effects of that release.

However, certain extensions of this work could generate substantial additional utility. For example, we may want to give access to a subset of the database – a non-random sample. This clearly will allow facts to be learned with respect to that subset, but can we use this approach to state limits on what can be learned that is not specific to that subset? Alternatively, can we give query access, but "cut off" access before too much is released. This requires tracking access over time, as a collection of small independent samples must be treated as a single large sample for our purposes. This is a problem faced with all inference protection mechanisms, Marks[11] solves this for "normal" inference with a mechanism for tracking and limiting what a given user has seen over time.

2 Motivating Example

In this section we will give a more detailed presentation of the "inference through supply orders" example, along with a "proof by example" of how providing only a sample of the data can be used to prevent data mining from making the inference *base SB is likely to support an SSA due to similar ordering patterns to bases X and Y.*

First, let us assume that the complete order database is as presented in Table 1. If we were to look for a classification of *Item* in this table, we would find the rules:

> If Location=X then Item=Fuel (confidence 100%, support 29%)
> If Location=Y then Item=Fuel (confidence 100%, support 14%)
> If Location=SB then Item=Fuel (confidence 100%, support 29%)

Armed with this knowledge, we can search for reasons why X and Y order only Fuel (other reasons that differentiate them from A and B). If we find a common factor (e.g., they support the SSA), we can make a good guess that SB also has this common factor.

Note that if we only have a sample of the database, we may develop a different set of rules. Table 2 gives a database containing 70% of the complete database. The rules generated from this database are:

> If Location=A then Item=Fuel (confidence 100%, support 17%)
> If Location=X then Item=Fuel (confidence 100%, support 33%)
> If Location=SB then Item=Fuel (confidence 100%, support 33%)

If we looked for common factors between Locations A and X (that differentiated them from others), we would not find the "threatening" evidence that they both support the SSA. Thus we have preserved the secrecy of the SSA at SB.

Note that there are a number of problems with this example:

1. Note that the support of the first rule is low. If the adversary were to ignore rules with support below 20% (in both cases), it might still be possible to obtain the desired information (although with less confidence, as the presence of the base Y

Table 1: Base Order Database

Date	Location	Item
1/1/97	SB	Fuel
1/1/97	X	Fuel
1/2/97	Y	Fuel
1/4/97	A	Fuel
1/6/97	B	Food
1/10/97	B	Food
1/18/97	A	Food
1/22/97	A	Food
1/24/97	B	Fuel
1/31/97	X	Fuel
2/3/97	SB	Fuel

Table 2: Sample of Base Order Database

Date	Location	Item
1/1/97	SB	Fuel
1/1/97	X	Fuel
1/4/97	A	Fuel
1/6/97	B	Food
1/10/97	B	Food
1/24/97	B	Fuel
1/31/97	X	Fuel
2/3/97	SB	Fuel

supporting the SSA, but not present in the rule, would lessen the impact of this).

2. What if we chose a different sample? It is easy to find samples that would *improve* the rules, and increase the adversary's ability to find the desired information.

3. Does this sample database still provide the desired information (e.g., an audit of order deliveries, or supporting the improvement of logistics through better prediction of ordering needs)?

Problem 3 is difficult – we must know the intended purpose of the data to determine if the sample is sufficient. Some purposes (such as prediction of ordering needs) are particularly problematic: If we aim to prevent an adversary from learning rules *we don't even know about*, we will necessarily prevent learning any rules. However, if the goal relies on specific data items, rather than inferences among the data items (such as comparing specific orders with their delivery traces), we need only ensure that the desired items are provided. We must be careful, though, to avoid problem 2 by letting the adversary choose the sample. Although not the focus of this paper, we will return to this issue in the conclusions.

What we address is problem 1. If we can show that the rules with high support or high confidence *on the sample* do not necessarily have high support and confidence *on the complete database*, then the adversary cannot rely on the rules produced from the sample. It is our goal to show not that the rules obtained on the sample *are* bad, but that they are *likely* to be bad. The sample may or may not cause unreliable results. If the adversary knows this, the confidence in the rules produced (and in any knowledge gained using those rules) becomes suspect. All we have to do is lower the adversary's confidence in the results to the point where the adversary would view any information gained from data mining on the sample to not be worth the effort required to verify it. In other words, although any knowledge gained using the rules may be correct, it may also be incorrect and thus cannot be trusted.

The goal of this work is to show that we can convince the adversary of the likelihood of such a failure, *without knowing the problem the adversary wants to solve.* Many data mining techniques (including the production rules shown above) produce rules or knowledge that can be used to classify items in the database. Vladimir Vapnik has shown error limits in classification when the classification is learned from a sample[14]. In Section 4, we will use an adaptation of Vapnik's work to show the difficulty of learning as a function of sample size.

To give an idea of how this would work in practice, we will use the results shown in the next two sections to demonstrate how large a sample would be reasonable for the above example.

Assuming there are at most two food and/or fuel orders per month, and that the adversary attempts to predict using the number of orders of each type per month, e.g., a rule is of the form:

$$January(Fuel, 2)\&January(Food, 0)\& \\ February(\ldots \Rightarrow \text{Supports SSA}$$

Further assume that using a collection of such rules we can develop a "perfect" classifier (one that will always give us the right result). If we are willing to tolerate that with a 50% probability, the learned classifier can be expected to be wrong 40% of the time, we can allow a sample size of over 175 billion where a single sample is all of the orders for a given base for a year (based on Theorem 1; we will discuss how these numbers are derived in Section 4.1). This is large, but the reason is that there are a great many possible rules (3 possible values for each of food and fuel gives 9 possible values per month; or 9^{12} possible values a year). If the sample doesn't contain an *exact* instance of a rule, the classifier won't know if it applies. Thus the need for a large sample size. This is a problem with complex classifiers – they don't generalize well.

The problem for the adversary is that there are likely to be too many ways to classify any sample – we are assuming that the adversary will be looking for rules based on *exact numbers* of each type of order. If we assume that a simple correct classifier exists, and that the adversary has the sense to look for a simple classifier, we have more serious constraints. In particular, if we assume N (the number of possible classifiers) is 6:

1. fuel >> food, low total order for the year
2. fuel $\approx$ food, low total order
3. fuel << food, low total order
4. fuel >> food, high total order
5. fuel $\approx$ food, high total order
6. fuel << food, high total order

we can only allow a sample size of $(6-1)/(4*.4) = 3$ (again, a sample is complete information on a base for a year, or 72 tuples from a complete version of Table 1.)

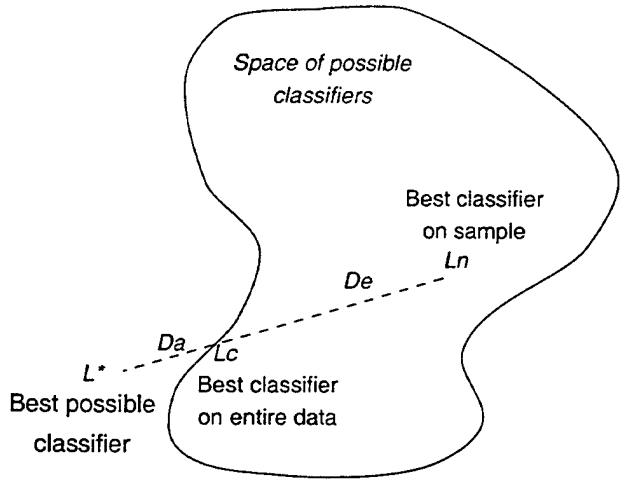


Figure 1: Distance between best classifier on sample and best classifier on data.

This assumes a perfect classifier exists, however with a simple classifier it is unlikely that it *can* perfectly classify the data. If the best possible classifier has some error, it is more difficult to state exactly what we mean by the error of the classifier learned from the sample. We will now give some more detail on error estimation. In Section 4 we discuss the specific theorems that allow us to determine these bounds; we will then return to discussion of this example.

3 Basic Ideas of Error Estimation

Our purpose in this section is to show how we can control the expected error of a classifier by limiting sample size. Figure 1 gives an idea of the problem. We want to control (by varying the sample size) the error D between the classifier the adversary can expect to learn from the sample and the "best possible" (Bayes) classifier. Note that the space of possible classifiers $\mathcal{C}$ may not include the Bayes classifier. Therefore the error is composed of two components: The approximation error D_a between the best classifier $L_{\mathcal{C}}$ available in $\mathcal{C}$ and the Bayes classifier L^*, and the estimation error D_e between the classifier L_n learned from the sample and $L_{\mathcal{C}}$. The primary difficulty is that there are many possible "best classifiers" L_n depending on the sample. Thus our goal is to analyze the *expected value* $\mathrm{E}\{D\}$ of the error with respect to the sample size.

There are actually three types of error:

Bayes Error: This is the expected error of the "best possible" classifier on the entire database. If the output is a function of the input, this is 0. This is entirely dependent on the data, and as such we can say little without knowing specifically what we want to protect against.

Approximation Error: This is the difference between the expected error of the best classifier from the types of classifiers we are considering, e.g., decision trees, and the Bayes classifier. The more complex the classifier, the lower the expected approximation error.

Estimation Error: This is the difference in expected error between a classifier learned from a sample and the best classifier available. This is what we can control by varying the sample size.

Various theorems show that classifiers exist that will learn "perfectly" given a sufficiently large sample, i.e.,

$$\mathrm{E}L_n \to L^*.$$

However, the classifier chosen may be dependent on the data or on n, for example this holds for a nearest neighbor classifier if the number of classes $k \to \infty$ and $k/n \to 0$ ([12]).

There are various things we might like to say:

1. Given an error estimate, that error estimate will be off (different from the Bayes error) by some amount ϵ with probability δ.
2. The expected difference between a learned classifier and the Bayes error is at least ϵ with probability δ.
3. Given a sample, the error of the learned classifier can be expected to differ from the best classifier *of that type* by ϵ with probability δ.
4. Given a type of classifier, we can expect the best possible classifier of that type to differ from the Bayes error by ϵ with probability δ.

Item 1 gives a lower bound – it says that even if the adversary "guesses" a great classifier, the adversary's estimate of how good the classifier is (using the sample) will likely be off. However, this is not likely to be a tight bound on the actual problem: what the adversary can learn. Likewise 4 isn't all that interesting; it says how good a classifier is possible, but nothing about what can be learned from the sample.

We will address 3, the estimation error. Figure 2 gives an idea of the goal: given a circle of radius ϵ, we want to choose a sample size such that with probability δ, the best classifier on the sample L_n will be outside the circle. If L_n is outside the circle, then at least ϵ percent of the time L_n will give the "wrong"

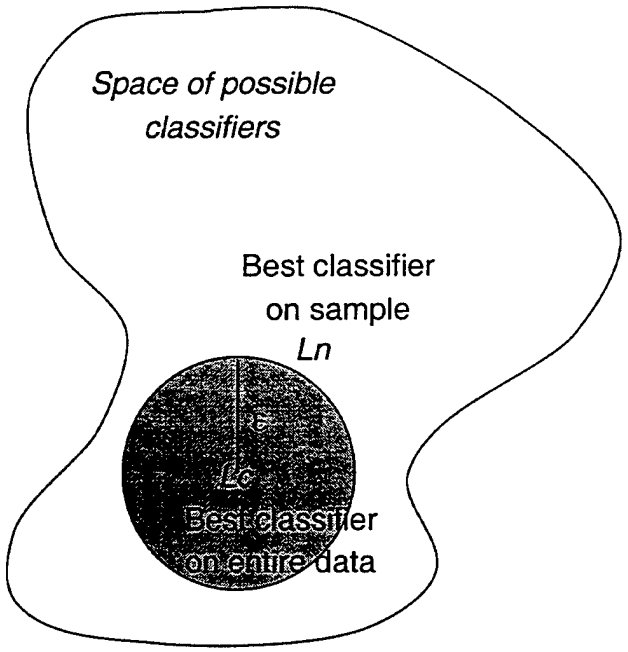


Figure 2: Error of best classifier on sample L_n worse than best classifier L_C by more than ϵ.

answer, even though a classifier L_C exists that would give the right answer.

We will also see that the formulas for estimation error are dependent on approximation error, giving a way of estimating 2.

4 Limits based on Sample Complexity

The *sample complexity* of a problem is the size of sample needed to ensure that the expected error of a classifier learned from the sample is within given bounds. The probability that a classifier is outside the given bounds allows us to look at two issues:

1. The likelihood that a different outcome is better for the given left-hand side of the rule, and
2. The likelihood that a better rule exists for the same outcome; i.e., a better predictor for the same result (lack of a good rule for the given outcome, bases that support the SSA, is the benefit gained from a small sample in Section 2).

What we can show is the following: Given a "best error" that we are willing to tolerate, and a minimum probability that the adversary will not be able to do better than that error, what is the largest sample we can allow? In this case, "best error" is a measure of the difference between the classifier (rule set) the adversary chooses, and the best possible classifier *of that type.* Note that this is dependent on the type of classifier; i.e., for a very simple classifier it is easy to get the "best" one (but the best one probably won't be very good). What we are bounding is the estimation error.

Formally, we are determining the sample complexity $N(\epsilon, \delta)$, defined as the smallest integer n such that

$$\sup_{(X,Y)\in\mathcal{X}} \mathbf{P}\{L(g_n) - L_C \geq \epsilon\} \leq \delta$$

where L_C is the best classifier in C:

$$L_C \stackrel{\text{def}}{=} \inf_{\phi\in C} \mathbf{P}\{\phi(X) \neq Y\}.$$

and $L(g_n)$ is the classifier selected based on the training data:

$$L(g_n) = \mathbf{P}\{g_n(X) \neq Y | ((X_1, Y_1), \ldots, (X_n, Y_n))\}$$

This states that there is a distribution of the data such that if the sample size $n < N(\epsilon, \delta)$, then with probability at least δ, a classifier exists that outperforms the chosen one by at least ϵ. The key factor here is that there is *a distribution* of the data where this holds. This doesn't mean a specific choice of the sample is necessary; the dependence is instead on the characteristics of the data as a whole. There exists a distribution of the data such that the bounds will hold on average across all random samples of the data.[1]

We begin with two definitions of Vapnik-Chervonenkis theory[14] (notation from [6]): First we define the *shatter coefficient* of the classifier:

Definition 1 *Let $\mathcal{A}$ be a collection of measurable sets. For $(z_1, \ldots, z_n) \in \{\Re^d\}^n$, let $N_{\mathcal{A}}(z_1, \ldots, z_n)$ be the number of different sets in*

$$\{\{z_1, \ldots, z_n\} \bigcap A; A \in \mathcal{A}\}.$$

The n-th shatter coefficient of $\mathcal{A}$ is

$$s(\mathcal{A}, n) = \max_{(z_1,\ldots,z_n)\in\{\Re^d\}^n} N_{\mathcal{A}}(z_1, \ldots, z_n).$$

That is, the shatter coefficient is the maximal number of different subsets of n points that can be picked out by the class of sets $\mathcal{A}$.

[1] One such distribution is skewed based on the classifier: most of the data conforms to a single rule left-hand side. This is not an unreasonable distribution to expect in practice. For example, if we assume that the information leading to a sensitive inference is a relatively small part of the database, we are likely to have this type of distribution.

Definition 2 *Let $\mathcal{A}$ be a collection of sets with $|\mathcal{A}| \geq 2$. The largest integer $k \geq 1$ for which $s(\mathcal{A}, k) = 2^k$ is denoted by $V_{\mathcal{A}}$, and it is called the Vapnik-Chervonenkis dimension (or VC dimension) of the class $\mathcal{A}$. If $s(\mathcal{A}, n) = 2^n$ for all n, then by definition, $V_{\mathcal{A}} = \infty$.*

We can see that the *shatter coefficient* of the classifier must be less than or equal to the number of distinct rule sets $= 2^{number\ of\ rules}$.[2] Since the VC-dimension is the largest k such that the shatter coefficient $= 2^k$, we can see that the VC-dimension for binary decision rules is the number of distinct rules.[3] Looking back at the example of Section 2, if we say that we are trying to learn if a base supports the SSA based on the number of fuel and food orders in a year, and we assume at most two orders of each type per month, we can see that for a year there are $24 * 24$ possible rules, so the VC dimension $= 576$. If we were to allow a more complex classifier, say the number of orders per month for each month (e.g., this would be useful if the SSA only flew in good weather), we would have (possible food orders per month $*$ possible fuel orders per month)$^{\text{number of months}} = 16777216$ possible rules.

We address this in two separate cases: Where a perfect classifier exists in $\mathcal{C}$, and where one does not. In the first case, what we are bounding is the total error (since there is no approximation error). There is a formula for this that holds for all $\delta \leq 1/2$:

Theorem 1 *[7, 6]. Let $\mathcal{C}$ be a class of discrimination functions with VC dimension $V \geq 2$. Let $\mathcal{X}$ be the set of all random variables (X, Y) for which $L_{\mathcal{C}} = 0$. For $\delta \leq 1/2$ and $\epsilon < 1/2$,*

$$N(\epsilon, \delta) \geq \frac{V-1}{4\epsilon}.$$

What this comes down to is the following. If a perfect classifier exists, and we have seen an example to which a rule applies, then we will always get that rule right. If we are asked to classify something where the training data didn't contain a similar sample (similar in the sense that a rule left-hand side matches), we will just be guessing. Thus, as the number of rules (V) goes up, the sample size needed does as well.

Figure 3 shows the minimum n needed for various values of ϵ and V. Note that the high values of V are likely to be more relevant in practice, as it is unlikely a perfect classifier will exist if the classifier is simple.

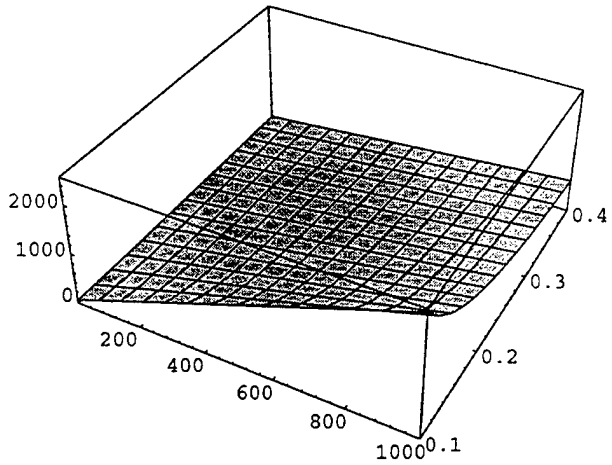


Figure 3: Value of n (vertical scale) where probability $\delta = 1/2$ that there is error ϵ, as function of error V (left scale) and ϵ (right scale), when a perfect classifier is available.

More interesting is what happens when there isn't a perfect classifier (the approximation error is greater than 0).

Theorem 2 *[7, 6]. Let $\mathcal{C}$ be a class of discrimination functions with VC dimension $V \geq 2$. Let $\mathcal{X}$ be the set of all random variables (X, Y) for which for fixed $L \in (0, 1/2)$,*

$$L = \inf_{g \in \mathcal{C}} \mathbf{P}\{g(X) \neq Y\}.$$

Then for every discrimination rule g_n based on $X_1, Y_1, \ldots, X_n, Y_n$,

$$N(\epsilon, \delta) \geq \frac{L(V-1)e^{-10}}{32} \times \min\left(\frac{1}{\delta^2}, \frac{1}{\epsilon^2}\right),$$

and also, for $\epsilon \leq L \leq 1/4$,

$$N(\epsilon, \delta) \geq \frac{L}{4\epsilon^2} \log \frac{1}{4\delta}.$$

Note that this gives us two bounds. One is dependent on L, V, ϵ, and δ (although for practical purposes we would let $\epsilon = \delta$ when using this); the second is only dependent on L, ϵ, and δ.

Some sample values for the first, based on $\epsilon = \delta = .1$ (10% of being wrong at least 10% of the time) are given in Figure 4. Intuitively, this is based on the probability of choosing the wrong rule to use; this

[2] Actually $m^{number\ of\ rules}$, where m is the number of possible output values.

[3] For non-binary, it works out to k s.t. $2^k = m^{number\ of\ rules}$, or $k = \log_2(m^{number\ of\ rules}) = number\ of\ rules \log_2(m)$.

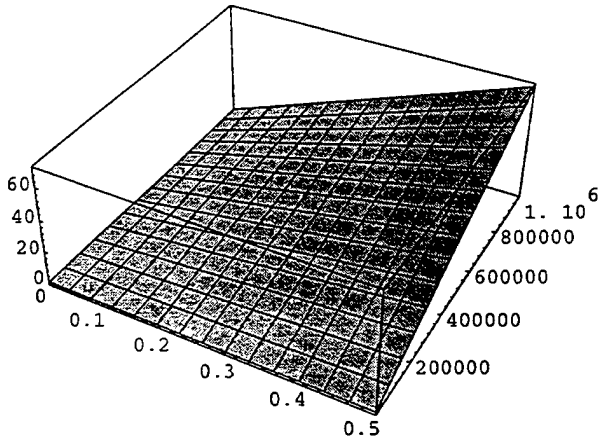


Figure 4: Value of n (vertical scale) below which error $\epsilon > .1$ with probability $\delta > .1$ as a function of L (left scale) and V (right scale).

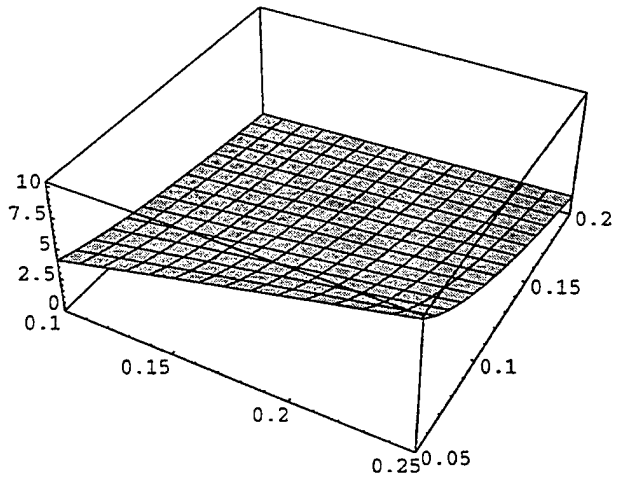


Figure 5: Value of n (vertical scale) below which a guarantee of error within 0.1 impossible as a function of L (left scale) and d (right scale).

gives a sample size if our primary concern is how the adversary chooses a given outcome rather than their ability to predict the outcome. In other words, the knowledge is in the rule, not in the application of the rule.

The second formula is intuitively based on guessing the wrong outcome for a given rule. Sample values are given in Figures 5 ($\epsilon \geq 0.1$) and 6 ($\epsilon \geq 0.05$).

Note that all of these are rather small sample sizes. However, they allow us to make a strong statement No matter how good the adversary's data mining technology, there are circumstances under which they can *expect* the results to be poor.

4.1 Back to the example

The figures given in Section 2 are based on Theorem 1. This is appropriate for the complex classifier case (a perfect classifier is likely), and due to the huge VC-dimension (9^{12}) of such a classifier we end up with a sample size $N(.4, .5) = (9^{12} - 1)/(4 * .4) > 175$ billion.

However, the simple classifier had a VC-dimension of 6. This gives a sample size of 3 if a perfect classifier exists. Theorem 2 handles the case where no perfect classifier exists. The first formula depends on large VC-dimension V (it is really only useful when $V > 15,000$). However, the second form gives us something to work with. If we start by assuming that such a simple classifier can be correct at most 75% of the time ($L = .25$), and we want the adversary to be forced to accept an error of 10% (in other words, they can expect to be right only 65% of the time, even though they could do as well as 75%) with probability $\delta = 0.15$, gives us our allowed sample of 3 years of data.

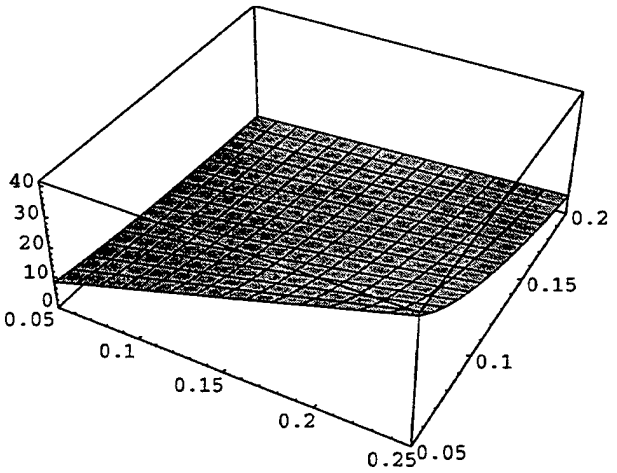


Figure 6: Value of n (vertical scale) below which a guarantee of error within 0.01 impossible as a function of L (left scale) and d (right scale).

Note that this is the same as the $L_C = 0$ (perfect classifier exists) case with $\epsilon = 0.4$ and $\delta = 0.5$. This seems strange; it appears easier to learn a good classifier when a perfect one doesn't exist. Intuitively, we can say that the *learning problem* is harder if a perfect answer exists.

4.2 Effect on a single rule

We have determined what the expected error is for a *set* of rules. The next question is, what confidence can the adversary have in a single inference rule?

The preceding section gives an answer to this question. Since what we have determined is the probability of the learned classifier failing to perform as well as the best possible classifier *on a given input*, it follows that a failure means that the given rule gives the wrong output. Thus it is the probability that for any given rule left-hand side (input), the output is "backwards" (since this is a binary classifier).

This, in a sense, is a worst case error: We have a rule that gives exactly the opposite of the proper result. Although the probability of this happening may seem small (.05 or .1), the result is still significant.

4.3 Non-binary outputs

Another interesting situation is what happens with multiple output categories. So far we have only discussed binary classifiers; what if their are more than two possible outcomes?

A simple way to model k categories is as $\log_2(k)$ binary classifiers (assuming the output categories are independent). The "combined" classifier will fail if *any* of the binary classifiers fail. Thus we can model the probability as

$$\begin{aligned}\mathbf{P}\{k\} &= \mathbf{P}\{k_1, \ldots, k_{\log_2(k)}\} \\ &= \mathbf{P}\{k_1\}\mathbf{P}\{k_2\}\cdots\mathbf{P}\{k_{\log_2(k)}\} \\ &\quad \text{(as the result "bits" are independent).} \\ &= \mathbf{P}\{\text{a binary classifier being correct}\}^{\log_2(k)}\end{aligned}$$

This is the probability of getting the result *correct*; or $1 - \mathbf{P}\{error\}$. Thus, if the expected error between the classifier (or rule) learned on the sample and the correct rule is .1, the chance of getting a rule right with 16 output categories is $\approx .66$. This is the probability of getting the same result as the best available classifier, which is also likely to have a larger error than in the binary case.

5 Unintentionally Released Data

A related problem is what happens if a sample is released? In other words, what if we know the sample size n? In this case, the goal is to determine "how bad" the loss is. What we can do is state limits on the probability that the error is within a given bound. In other words, we want to determine how confident the adversary can be in any result mined from the data.

This is similar to the results in the preceding section. Formally, the problem is to find lower bounds for

$$\sup \mathbf{E}L(g_n) - L_C.$$

What this states is that there is a distribution of the data where the adversary can expect they will be off by the given amount with a sample of size n randomly chosen over that distribution.

The following theorem gives us a way to make use of this information:

Theorem 3 *[13]: Let C be a class of discrimination functions with* VC *dimension V. Let $\mathcal{X}$ be the set of all random variables (X, Y) for which $L_C = 0$. Then, for every discrimination rule g_n based upon $X_1, Y_1, \ldots, X_n, Y_n$, and $n \geq V - 1$,*

$$\sup_{(X,Y)\in\mathcal{X}} \mathbf{E}L_n \geq \frac{V-1}{2en}\left(1 - \frac{1}{n}\right).$$

This says that if a perfect classifier exists, there is a distribution of the data such that any classifier learned from a random sample of the data will have at least expected error $\mathbf{E}L_n$, where this value is dependent on the VC-dimension V and the sample size n.

Since this function is continually decreasing for $n > 2$, it gives the maximum at the lower limit where the theorem applies: $n = V - 1$. At this point the upper bound is

$$\frac{1}{2e}(1 - \frac{1}{V}) \geq 0.18 \text{ when } V > 50.$$

This means that given a sample of size $n \leq V - 1$, it is possible that any classifier learned from the sample will be wrong 18% of the time (there exists a distribution of the data such that this will hold).

There are similar theorem for the case $V > 0$, but they only apply for large n, where the expected error is so small as to be nearly useless (less than 1%).

6 Conclusions and Further Work

Pattern recognition theory gives us tools to deal with security and privacy issues in data mining. Limiting the sample size that can be mined allows us to state clear limits on what can be learned from the sample. These limits are in the form of expected error on what is learned. What they allow us to do is tell an adversary, "Here is a sample you may mine, but you can expect any result you get will be wrong ϵ% of the time with probability δ, *no matter how good your*

data mining is". It gives us sample sizes where we can *expect* that the sample may be misleading.

One advantage of this approach is that the *method* can be open. The adversary cannot use knowledge of how we restrict sample size to improve the data mining process. In fact, the knowledge that results from mining the sample cannot be trusted may discourage the adversary from making the attempt.

These sample sizes tend to be small (10s or 100s of tuples). However, for certain purposes this is reasonable. For example, providing samples of actual data to be used for development of new systems to operate in a secured environment. These formulas give us the ability to state "this is a safe amount of data to release", without worrying about the *specific* inferences that may be drawn. This is independent of the external knowledge available to the adversary (except for the database contents not included in the sample).

Another thing we gain is the ability to analyze the effect of a given sample size. This is useful when data is released unintentionally; we can analyze the potential impact of the release both in terms of the direct inferences that can be made, and the "probabilistic inferences" that can be determined by data mining. Again, this is independent of the technology or external knowledge available to the adversary.

There are various ways we can extend this work. One is in the realm of support to a data security administrator; an "operations manual" for determining how much data to release. The primary effort required is determining appropriate parameters for a classifier. One solution would be to use clustering techniques on the database (e.g., self-organizing maps [10] with thresholds on nearest neighbor) to give a likely value for the VC-dimension of a reasonable classifier. This idea is based on grouping the potential rule left-hand sides into similar groups, with the idea that similar inputs would likely lead to similar outputs. A classifier on extremely diverse data is likely to be more complex than one on simple data. This needs more work to establish limits on the probabilities involved.

Another area for extension is that this work assumes the sample obtained by the adversary is randomly distributed. However, many applications will produce non-random samples. An example of this would be a system that allows queries to a database (access to individual information), but tries to protect correlations among items through limiting the volume of data available. In such a system *the adversary controls the sample distribution.* What can we say about such an environment?

There are a number of possibilities:

- The sample is random with respect to a correlation discovered. In this case, the fact that the sample is not random with respect to *some* criteria is irrelevant.

- A discovered correlation involves the selection criteria. The problem is that we cannot say if the correlation is *unique* to the selection criteria: It may or may not be independent of the selection criteria.

- A correlation exists between the selection criteria and some other field in the data. The previous case prevents our discovering this correlation, however does the non-randomness of the sample allow us to discover *other* correlations between the "other field" and other items? Or does this reduce to the previous case?

- The adversary has multiple samples based on different selection criteria. One obvious sub-case of this is a random sample and a non-random sample. Does this allow us to discover correlations with respect to the selection criteria that we would not expect to discover with a random sample? As a worst case, this would give us the effect of a sample size as large as the size of a random sample required to give all the selected items. As a best case, this would appear as a random sample. The actual bound is probably somewhere in the middle – this needs to be worked out.

This is related to work in privacy problems from data aggregation [3, 1]. The statistical database inference problem deals with identifying individual data values from one or more summary queries. In a sense it is the converse of the problem of this paper; instead of protecting against learning data items from aggregates, we are protecting against learning aggregates from individual items. Although the basic problem is quite different, as we move toward non-random samples the two areas may overlap. Of particular note is work on random sampling queries [5]; this may provide tools to implement policies governing the creation of non-random samples.

Another possible starting point for this is artificial intelligence work on selection of training data [2, 15]. Preventing the adversary from selecting a "good" set of training data (while still allowing some queries, and those non-random release of data) would support this work.

Another area is the effect on correlations, rather than inference rules. The example in Section 2 was

based on developing a *classifier* to predict based supporting the SSA. Alternatively, the correlation between "lots of fuel" and "SSA" may be of interest. The problem is similar, however understanding the effect of small samples on the significance of such correlations (e.g., chi-squared measures) is still open.

What we have shown is that for reasonably small samples, we can be confident that the threat posed by data mining is minor.

References

[1] Sumit Dutta Chowdhury, George T. Duncan, Ramayya Krishnan, Stephen Roehrig, and Sumitra Mukherjee. Logical vs. numerical inference on statistical databases. In *Proceedings of the Twenty-Ninth Hawaii International Conference on System Sciences*, pages 3–10, January 3–6 1996.

[2] Dawn M. Cohen, Casimir Kulikowski, and Helen Berman. DEXTER: A system that experiments with choices of training data using expert knowledge in the domain of DNA hydration. *Machine Learning*, 21:81–101, 1995.

[3] Lawrence H. Cox. Protecting confidentiality in small population health and environmental statistics. *Statistics in Medicine*, 15:1895–1905, 1996.

[4] Harry S. Delugach and Thomas H. Hinke. Wizard: A database inference analysis and detection system. *IEEE Transactions on Knowledge and Data Engineering*, 8(1), February 1996.

[5] Dorothy E. Denning. Secure statistical databases with random sample queries. *ACM Transactions on Database Systems*, 5(3):291–315, September 1980.

[6] Luc Devroye, László Györfi, and Gábor Lugosi. *A Probabilistic Theory of Pattern Recognition.* Springer-Verlag, New York, 1996.

[7] Luc Devroye and Gábor Lugosi. Lower bounds in pattern recognition and learning. *Pattern Recognition*, 28:1011–1018, 1995.

[8] Thomas H. Hinke and Harry S. Delugach. Aerie: An inference modeling and detection approach for databases. In Bhavani Thuraisingham and Carl Landwehr, editors, *Database Security, VI, Status and Prospects: Proceedings of the IFIP WG 11.3 Workshop on Database Security*, pages 179–193, Vancouver, Canada, August 19-21 1992. IFIP, Elsevier Science Publishers B.V. (North-Holland).

[9] Thomas H. Hinke, Harry S. Delugach, and Randall P. Wolf. Protecting databases from inference attacks. *Computers and Security*, 16(8):687–708, 1997.

[10] Teuvo Kohonen. The self organizing map. *IEEE Transactions on Computers*, 78(9):1464–1480, 1990.

[11] Donald G. Marks. Inference in MLS database systems. *IEEE Transactions on Knowledge and Data Engineering*, 8(1), February 1996.

[12] C. Stone. Consistent nonparametric regression. *Annals of Statistics*, (8):1348–1360, 1977.

[13] V. Vapnik and A. Chervonenkis. Theory of pattern recognition, 1974. (in Russian).

[14] Vladimir Naumovich Vapnik. *Estimation of dependences based on empirical data.* Springer-Verlag, New York, 1982.

[15] Jihoon Yang and Vasant Honavar. Feature subset selection using a genetic algorithm. *IEEE INTELLIGENT SYSTEMS*, 13(2):44–49, March/April 1998.

[16] R. Yip and K. Levitt. The design and implementation of a data level database inference detection system. In *Proceedings of the Twelfth Annual IFIP WG 11.3 Working Conference on Database Security*, July 15–17 1998.

Security Administration for Federations, Warehouses, and other Derived Data

Arnon Rosenthal, Edward Sciore, Vinti Doshi[1]

Abstract

Security administration is harder in databases that have multiple tiers of derived data, such as federations, warehouses, or systems with many views. Metadata (e.g., security requirements) expressed at each tier must be visible and understood at the other tier. We describe several use cases in which tiers negotiate to reconcile their business requirements. The sources must grant enough privileges for the derived tier to support the applications; the derived tier must enforce enough restrictions so that the sources' concerns are met; and the relationship between the privileges at source and derived tier must be visible and auditable.

The guiding principle is that a security policy should primarily govern *information*; controls over source tables and views from which the information can be obtained are intended to implement such policies. We require a kind of global consistency of policy, based on what information owners assert about tables and views. Deviations are primarily a local affair, and must occur within safe bounds. Our theory examines view definitions, and includes query rewrite rules, differing granularities, and permissions granted on views. Finally, we identify open problems for researchers and tool vendors.

1 Introduction

Federations and warehouses [Osz, Inm] are examples of *multi-tier* database systems. Data originates in a *source* tier. The *derived* tier consists of information derived from the sources. The system should have an integrated security policy, so that each granule of data is protected consistently, whether obtained through the source or derived tier. Creating such a policy requires negotiations between administrators at different tiers.

The goal of this paper is to alert the security community to the administration problems in such multi-tier systems, and to suggest a model within which the necessary research can be done. We present motivating scenarios, and give initial sketches of the necessary theory, methodologies and automated tool support. The metadata management problem for warehouses and federations is broad and murky. Our main contribution is to formulate a problem that can be fruitfully addressed both by tool vendors and by researchers.

1.1 The Need for Collaboration

The tiers in a multi-tier database are often administered separately, which makes security administration harder than in a centralized system. Administrative activities at one tier may affect other tiers, and thus need to be visible and understood at these tiers. For example, if a source tier grants or revokes the access permissions on one of its tables, it must communicate this to derived

[1] Addresses: A. Rosenthal, V. Doshi, The MITRE Corporation {arnie, vdoshi}@mitre.org.
E. Sciore, Boston College (and MITRE) sciore@bc.edu.

tiers that use that table.[2] Similarly, if a derived tier wishes to give a particular subject (e.g., user, role, group) access to one of its tables, it must request permission from the (source tier) owners of information used to derived the table, in terms of each owner's schemas.

The administrators of each tier must be able to *negotiate* with each other, in order to match security and business requirements. In particular, negotiation aims at:

- Protecting information, as its owners require.
- Giving sufficient access permissions so applications can accomplish their tasks.
- Enforcing the agreed-upon access policies as efficiently and reliably as possible.
- Allowing administrators at different tiers to understand each other's concerns, without needing to understand each other's schema and the derivation logic.

1.2 Source versus Derived Permissions

Administrators need to see all relevant security metadata, whether initially provided in terms of their "native" schema or a foreign schema. To enable this visibility, negotiation support must include a translation step. We sketch the theory here, and revisit in more detail in section 3.

Current multi-tier systems do not really connect the source-tier and derived-tier permissions. For example in a federation, a middleware product (as the owner of federation views) may be given unlimited Read access to sources, with Grant option. This approach places great burdens and trust on the federation administrator, but provides neither guidance nor tools. In particular, there is no formal basis for source administrators to judge whether the permissions that the federation administrator grants are appropriate.

Our approach is based on three premises:

1. Ownership and access controls are fundamentally about *information,* not physical artifacts (source tables) or interfaces (views). That is, protection must be global. This implies that access controls asserted at one schema must be *propagated* to the other schemas.

2. A table's permissions are the sum of two sources: permissions explicitly asserted on the table, plus permissions *inferred* from permissions on other tables. If you have Read permission on tables X and Y, you get permission on any view you can compute from X and Y. Conversely, one gets read permission for a view V if there is *any* query that computes V from tables to which you have access.

3. Administrators of implementation units (source tables, views, replicas) may reduce the permissions allowed by (2), but not expand them. These restrictions often stem from a desire to control physical resources. To protect response times or generate revenues, one database might access only by known subscribers. To reduce hacker threats, another might allow access only by the enterprise's employees.

These premises guarantee that the access permissions in a database are consistent. That is, if a user has access to information in one part of the database, the user has access to that information everywhere in the database. Section 3.6 discusses a mechanism for superimposing local restrictions.

---

[2] Henceforth, we shall use the terms "view" and "derived table" interchangeably, allowing virtual and materialized derived tables, which need not be perfectly consistent. "Table" comprises both source tables and views.

1.3 Assumptions and Simplifications

Our work does not address the usual problems of integrating distributed heterogeneous data, nor deal with all aspects of security for an enterprise database. We assume that a metadata management environment (denoted *MME*), provides the illusion of several kinds of uniformity:

- *Schemas, metadata, and distributed databases:* The source and view schemas and their metadata are represented in a repository. All schemas and views are in the same language (e.g., SQL).

- *Permission types:* MME defines a uniform set of permission types (e.g., Create, Read, Update, Delete), across all tiers.

- *Subjects to whom privileges are granted:* MME manages grants of access permissions to *subjects* (e.g., users or roles) that are globally known across tiers. Authentication, management of role membership, and subject globalization (i.e., specifying which global subject corresponds to each DBMS or operating system subject) are outside our scope.

Some further assumptions simplify our prose. We believe that each of them could easily be removed, as indicated:

- We speak in terms of only two tiers – a source tier and a derived tier – so each administrator sees just one "foreign" tier. (Actually, the derivation graph can be arbitrary.)

- We assume that view definitions are readable by all users(rather than allow them to be protected objects).

- We ignore delays in propagating data and metadata updates.

- We do not model Grants of Grant authority. Instead, we merely speak of owners of information. An owner can work in terms of source tables (the usual case) or view tables[3] (e.g., if the sources are merely an efficient physical representation of a natural conceptual table).

Federations and warehouses bring together a great deal of information, increasing the *aggregation vulnerabilities:* Information that in isolation was not sensitive can become sensitive when made accessible together, e.g., via a federation [Jaj, Thu]. However, the proposed solutions to this problem for centralized systems seem to provide modest extra security, at very high cost in run-time and administration (e.g., quadratic growth). We therefore do not address this issue.

1.4 Paper Roadmap

Section 2 covers the kinds of negotiation that are possible between the source and derived tiers. In it, we show that communication can be split into a requested action and a translation of permissions from one schema to another. Section 3 examines the underlying theory that allows

[3] To become owner of a view, one must somehow receive authorization from owners of all information used in forming the view. Mechanisms for such collaborative delegation are a subject for future research.

this translation to proceed. Section 4 discusses conclusions and suggests promising open research problems.

2 Negotiation Scenarios

The tiers in a multi-tier database may be administered separately, with conflicting goals. This section illustrates the kinds of negotiations that must be performed, and implicitly the kinds of capabilities a metadata coordination tool must provide. Its research contribution is to identify, from the overwhelming set of real requirements, scenarios that point at useful, feasible facilities for permission negotiation and translation. In particular, we try to answer:

- *What sort of negotiations do the tiers' administrators need to perform?*
- *What sorts of requests do they need to send to each other?* There appear to be a mix of commands, proposals, notifications, and comparisons.
- *How are requests for permissions translated to the recipients' schemas*? Each party needs to know what has been said, in terms of its own schema.

Many tasks involve a set of permissions against the requestor's schema, a target schema, a recipient for the permissions (translated to refer to the target schema), and a desired *action.* The action[4] may be a command to install the translated permissions ("Enforce these permissions at your tier."), a proposal ("I would like you to cause these permissions to be true."), a description of what the tier is doing ("For your information: Here are the permissions I enforce."), or a hypothetical display ("If I granted these permissions, how would it look on the target schema?")

The scenarios below present more detail. The following 2-tier Hospital database provides a running example. The source tier contains the two base tables:

PATIENT (Patient_Id, Insurance_Co)
PROCEDURE (Patient_Id, Procedure_Performed, Doctor, Bill_Amount, Date)

The view tier contains the derived tables:

DOCTOR_ACTIVITY (Doctor, Procedure_Performed, Month, Year)
INSURANCE (Insurance_Co, Month, Year, Total_Billed)

The first view is a selection from PROCEDURE; the second joins PATIENT and PROCEDURE, and then groups by Insurance_Co and (Month, Year) from Date, and totals the Bill_Amount.

2.1 Bottom-Up: Conforming to implied permissions

In the first scenario, the view tier is a data warehouse over the Hospital database. Derived tables are materialized views, whose derivation may involve complex fusion and scrubbing logic. The warehouse handles user requests without referring back to sources, and hence checks all permissions itself. The warehouse's permissions should conform to the source's intentions.

The source tier keeps the warehouse informed about what access permissions to enforce. Whenever a grant or revoke command occurs at the source, the metadata management environment transmits the command to the warehouse, and translates the source's command to an appropriate action on the appropriate view tables. For example, a grant on PROCEDURE

[4] These actions are themselves subject to permissions. A table owner may impose limits on who can enter requests that the responsible dba sees, and drastically limit those who can automatically install permissions.

translates to a grant on DOCTOR_ACTIVITY and, if there is an existing corresponding grant on PATIENT, a grant on INSURANCE.

Automated translation is crucial, both to reduce administrator workload and to keep the system secure in a dynamic environment. When a source revokes a permission, the warehouse should immediately reflect the revocation. A message expressed in terms of source tables will be unintelligible to installer scripts at the warehouse DBMS. Even human administrators will react slowly and unreliably when presented with changes in terms of another tier's tables.

In the second scenario, the derived tier is a federated database. Permissions are asserted against source schemas, and enforcement occurs after the user's query is translated to a query on source tables. The system can execute without translating permissions to refer to derived tables. But derived tier users want to use their own tables when inspecting the permissions that they possess, or when they have been refused access. Consequently, the source administrator should transmit grant and revoke permissions to the derived tier, as in scenario 1.

In the general case, some views may be derivable from other views. In such cases, permissions on the lower views will also need to propagate upward.

Our example views are simpler than the general case, but we believe such simplicity occurs often in real life. Even complex derivations often return some attributes that were simply pulled from source tables. For the remaining cases, we contend that an administrator's assistant need not be complete to be useful. In fact, it may be very helpful for researchers to point out and formalize easy cases, where there can be substantial benefit at little cost.

2.2 Top-Down: Proposing new permissions

This scenario applies to either the warehouse or federation examples above. Suppose the derived tier administrator proposes an increase in her users' permissions. For example, suppose cost analysts need to be able to read the INSURANCE table. The derived tier administrator sends a message to the source tier requesting this change.

The system executes this message as follows. The first step is to compare the desired and existing permissions on the derived tables, so that only the additional *(delta)* permissions will be requested. There should be less work in judging the delta, and we avoid redundant grants that invite confusion when privileges are revoked. In the example, suppose costAnalyst already has access to PROCEDURE, and requests access to view INSURANCE. When the request is translated to the sources, one would ask for a new Grant only on PATIENT. In systems with column permissions, one might determine that Analyst already had permission to read the first three INSURANCE columns, so the downward translation consists of a request for access only to INSURANCE.Billed_Amount.

At this point, the source administrator has several options. If she is willing to make the change, agreement has been reached. Or she may instead make a counterproposal. The counteroffer is translated in "bottom-up" mode to the derived tier administrator, who again may agree, or continue the negotiation.

A counterproposal may take several forms. The administrator may propose a different user (e.g. "costAnalystSupervisor") be assigned the desired permissions. Or the two administrators might jointly determine that a new user role is required.

Alternatively, the administrator may propose to create one or more views that are useful in answering derived tier queries, but are less sensitive than the underlying tables [Sto75]. For example, given a request for cost analyst access to the PATIENT table, the source administrator may counter-propose that cost analysts be given access to a view that contains only those patients who were seen this year. Or the source administrator, feeling that Patient_Id values must be kept confidential, gives analysts access to a view that is the join of PATIENT and PROCEDURE, projecting out the Patient_Id. The view query for INSURANCE can be rewritten to employ that view instead.[5]

Finally, we note that it may be useful for the source to grant a wider set of permissions than were requested. For example, suppose the derived tier administrator requests that cost analysts be given access to:

```
Select Procedure_Performed, Bill_Amount
From PATIENT
Where Year > 1996
```

The source administrator determines that this information is suitable for cost analysts to see, and therefore is willing to create a view for it. However, the administrator realizes that other fields are equally releasable (e.g., Doctor, Date), as are pre-1996 records. Furthermore, it is likely that cost analysts will later want to see other fields or selection criteria. Hence, the source administrator chooses to define and grant Read access to the following wider view:

```
Select Procedure_Performed, Bill_Amount, Doctor, Date
From PATIENT
```

Expanding the view contravenes the principle of least privilege, but may reduce the administrator's workload substantially.

In the above scenarios, the source had ultimate control over permissions. Our approach also accommodates scenarios where the security officer works in terms of the view schema. For example, often a conceptual object (e.g., HOSPITAL) is partitioned and denormalized for efficiency, and expressed as a view. The physical tables represent no natural application unit. For administering information security, the HOSPITAL view may be a more natural venue..

2.3 Comparison: Check the consistency of the two tiers

If all tiers faithfully imposed the information owners' permissions, the permissions would all be *consistent.* This is not how systems are administered today. For example, suppose information owners use just the source schema to express permissions, and consider a warehouse that enforces permissions on queries against the (derived) warehouse schema. Auditors may want to check whether all permissions granted on the warehouse can be inferred from the source permissions. If not, there may be a serious security violation. Today, such checks are so troublesome that they are rarely done. With automated comparisons, one could do a nightly consistency check on critical information.

All of these comparisons require that permissions be expressed in terms of the same schema. One can translate source permissions upward, to refer to the derived schema. These are easily compared with the permissions actually granted by a warehouse. The resulting exception report identifies both warehouse permissions that were not justified by sources, and source permissions

[5] The view need not be rewritten if the system takes advantage of query equivalence, as discussed in Section 3.5.

that the warehouse has chosen not to grant. If desired, the exception report can be treated as a proposal, to be propagated downward (as discussed in section 2.2).

2.4 Pointers to Additional Material

A mockup demonstration of elementary negotiation and translation is available on the web pages [Ros99b]. An interface mockup shows administrators communicating proposed metadata changes upwards and downwards. It also shows the rules for translating permissions (and some other kinds of metadata) to refer to the other tier's schema, and the management of these translation rules. A paper describing more sophisticated negotiation support is also available there. Frameworks for building and extending such tools are discussed in [Ros98, Ros99a].

3 Handling Access Control Metadata

The scenarios above illustrated database administrator's requirements in a multi-tier database, especially for negotiations. This section focuses on the model of permissions, and their translation (i.e., inference) across tiers. To encourage vendors to move the results into product, we seek broad applicability but simple implementation.

3.1 Preliminaries

Access permissions are associated with source or view tables in a schema. There are two popular ways of representing a table's access permissions: as a set of Access Control Lists (i.e., {subjects with a given permission}), or as a single access predicate. A predicate formalism is more flexible (it lets us include other tests, e.g., on time-of-day or user's department) and has a solid base in logic. The ACL notation is more intuitive for Granting and Revoking. We use access predicates, but the same results hold for ACLs.

Formally, the expression *Allow(T, p)* denotes that if predicate p is satisfied, one can access table T[6]. A table can have any number of such access permissions with "OR" semantics, e.g., from separate SQL Grant operations.

We separate a table's permissions into

- *Information* permissions: Here, the information's owner wishes to impose controls on all routes through which a particular kind of information (e.g., patients' names) can be accessed. From this information policy, one derives permissions to be allowed on each table.
- *Physical resource* permissions: Permissions that are associated with concrete processing resources (e.g., physical tables, interfaces with executable code). These allow administrators to deal with local requirements for system security, performance, or payment.

To access information from a table, one needs *both* the information and the physical resource permissions. Administrators working with derived tables will need answers about both, e.g., "Are cost-analysts allowed to read patient names" and "Will some system that possesses patient names allow me to run my queries?" The two properties are likely to be administered separately. Information permissions are discussed in sections 3.1-3.4, and physical resources in 3.5.

[6] To simplify notation, we omit "subject" and "operation" as explicit arguments. Until the last part of section 3.2, our examples focus on Read.

3.2 Inferring Access Permissions

Given a (base or derived) table T, the following rules define the inferred permissions associated with T.

All_perms(T)	= OR{ Direct_perms(T), Inferred_perms(T) }
Direct_perms(T)	= OR{ p | Allow(T, p) was asserted }
Inferred_perms(T)	= OR{ Query_perms(T,Qi) | T can be computed by query Qi }
Query_perms(T,Q)	= AND{ All_perms (Ti) | query Q mentions table Ti }

The first two rules are straightforward. The first rule states that a permission on T can be either directly asserted or inferred. The second rule formalizes the idea (from section 3.1) that the meaning of multiple direct assertions is the same as the OR of the individual predicates.

The last two rules go together, to say that you get access permission on a table T if you have (direct or inferred) permission on all tables mentioned by some query that calculates T. To illustrate, consider a typical case of inferring permissions on a view table V defined by query Q(T1,...,Tk). Then V (by definition) can be computed by Q. Assuming no other query computes V, the rules state that

Inferred_perms(V) = AND{All_perms(T1),...,All_perms(Tk)}.

In other words, anyone who has access permissions on Q can be inferred to have the same permissions on V.

One might also infer permissions on base tables. For example, consider the base table PROCEDURE from Section 2. Suppose the following views were defined:

```
Create view V1 as   select * from PROCEDURE
                    where Bill_Amount > 1000;
Create view V2 as   select * from PROCEDURE
                    where Bill_Amount <= 1000 or Bill_Amount is null;
```

If a user has an access permission on both V1 and V2, then this same permission can be inferred for PROCEDURE.

The key concept in the above rules is: Any query that can compute T can be used to infer permissions on T. This of course begs the question of how such queries can be found. This issue is sufficiently important that we devote Section 3.3 to it.

Our approach also handles update operations on views, whose implementation is expressed as a sequence of source-table operations.[7] In all cases, the view update is replaced by a sequence of database operations (i.e., a procedure) that carries out the desired update. We can easily adapt the

7 Current database tools (e.g., Oracle's CASE product) handle many cases of view update, with a variety of mechanisms. Depending on tools and performance issues, the desired semantics can be specified as a stored procedure or method, as a trigger, or as a metadata-driven request translator in the development environment or user interface. Triggered updates may execute under the definer's permissions, which can be checked when the trigger is compiled.

inference rules to such situations, by replacing "T can be computed by query Qi" with "T can be implemented by a procedure Pi".

Note that the generated operations need not all be the same flavor as the original. For example, a request to delete from a join view V = R1 join R2 join R3 might translate to "Delete from R1; Delete from R2; Read R3 to check integrity, and Abort if violated". The access permission for the view update is the AND of permissions for the two Deletions and the Read.

3.3 Imperfect Solutions to the Inference / Rewrite Problem

The inference rules of Section 3.2 define the allowable access permissions on a table T by referring to the set of all queries that can compute T. But this set may be neither robust (e.g., new types of integrity constraints will require more powerful inference), nor feasible to enumerate.

Our approach is to recognize that the inference rules can be understood both theoretically and pragmatically. The *theoretical mode* considers the set of all queries that are mathematically equivalent. The *pragmatic mode* considers the set of equivalent queries the system finds feasible to generate.

The theoretical mode is sound (in the sense of allowing only rewrite-justified permissions) and complete (since by definition, it considers all rewrites). But it is no basis for building a system. If the constraint theory is rich, the problem of finding all queries equivalent to some Q is undecidable; at intermediate richness, it still may be computationally prohibitive. Also, completeness is fragile as systems evolve— if one introduces a new type of constraint, the rewrite process may no longer be complete.

The pragmatic mode is sound[8], but not complete. A pragmatic access-permission checker does its best to identify a rewrite that enables the query to execute, but does not guarantee to find it. It dodges both the semantic and exponential-growth problems. We argue that pragmatic mode is the right basis for building a system. There is ample precedent for DBMS facilities that are sound but may fail. For example, some SQL queries will timeout or crash due to huge temporaries – thus the set of executable queries depends on the cleverness of the query processor.

The difference between theoretical and pragmatic mode only involves inferred permissions. The pragmatic mode would begin with the explicit permissions (rules 1 and 2), and then heuristically infer additional ones; hence it gives at least the permissions of current SQL systems. Any additional inferred permissions can be considered a convenience to the user. And a user who is cleverer than the automated search can do the replacement manually.

A final advantage of our approach is that we can take advantage of query-processing technology. Query processors are a critical, large (> 100K lines of code) well-funded part of DBMS implementations. By using the rewrite set generated by the query processor, the security subsystem can improve as the query optimizer improves.

3.4 Exploiting Rewrites

This section explores the use of query rewrite in security semantics.

[8] This observation results from the fact that the inferred access predicate is the OR over the set of rewrites. Because pragmatic mode considers fewer rewrites, the inferred predicate will be less general.

3.4.1 View Substitution

View substitution is the process of replacing a mention of a view in a query by the view's definition [Sto75]. For example, the following query mentions the view INSURANCE:

```
select Insurance_Co, Total_Billed
from INSURANCE
where Year = 1999
```

It can be rewritten to use only source tables:

```
select p.Insurance_Co, sum(r.Bill_Amount) as Total_Billed
from PATIENT p, PROCEDURE r
where p.Patient_Id = r.Patient_Id
group by Insurance_Co, yearof(r.Date)
having yearof(r.Date) = 1999
```

Our approach gives the option of view definition without separate security administration. To users who have the necessary permissions on the source tables, the view is simply an alternate interface with no security ramifications. In contrast, the current SQL standard requires that view users obtain explicit grants from the view owner.

We recognize that sometimes executing a view query $Q_V(T_1, \ldots, T_n)$ adds knowledge, making the result more sensitive than its inputs[9]. We cannot solve the entire aggregation problem, but can protect knowledge embodied in the view definition. This requires no change to our model – one simply protects the view text and (possibly with looser permissions) its executable.

3.4.2 Constraint-based Simplifications

If the user queries a derived table V, some source data that underlies V may be irrelevant to the query result. Query processors routinely exploit integrity constraints and relational algebra to rewrite queries in a simpler form. In terms of our inference rules, a query needs only permissions for the data it accesses (using the simplified expression). Such inference can substantially increase the set of view queries that users are allowed to execute.

The following example illustrates the benefit. Suppose the source database has a foreign key constraint on Patient_Id (i.e., every record in PROCEDURE has a unique corresponding PATIENT). Suppose also that the derived tier contains a table that is the join of the two source tables, as in:

```
Create view MED_INFO as
Select p.*, r.*
From PATIENT p, PROCEDURE r
Where p.Patient_Id = r.Patient_Id
```

Suppose a user issues the following query Q1 on the derived table:

```
Select Patient_Id, Procedure_Performed
From MED_INFO
```

[9] The other side of the aggregation problem is to prevent two items from being revealed together. Commercial DBMSs offer no protection on this score, nor are they likely to. A user is always free to type in a query that retrieves the information directly from source tables, in one or multiple queries.

A straightforward evaluation of Q1 requires permissions on both PATIENT and PROCEDURE, or on the entire view MED_INFO. But note that the selected attributes all derive from PROCEDURE, and (by the foreign key constraint) no PROCEDURE records drop out of the join. A query processor can rewrite Q1 to access only PROCEDURE, so permissions on PATIENT are unnecessary.

Suppose now that a user issues an analogous query Q2 for PATIENT attributes:

```
Select Insurance_Co
From MED_INFO
```

The simplified query might be:

```
Select Insurance_Co
From PATIENT
Where Patient_Id in (Select Patient_Id From PROCEDURE)
```

The query can thus be executed if the source provides permissions on the Patient_Id column of PROCEDURE (either through column granularity or through a view).

If one could define methods, one could get by with even less access. Frequently a company will allow outsiders to request a single phone number, but not to download a telephone directory. Analogously, we don't need full permissions on the PROCEDURE table; we just need to know whether a given patient appears in the table. (Oracle's "reference" privilege is somewhat similar, but executes under the constraint-definer's privileges). Let isPresent(value, column) denote a method that returns TRUE if the value appears in the specified column. Query Q2 could then be rewritten as:

```
Select Insurance_Co
From PATIENT
Where isPresent(Patient_Id, PROCEDURE.Patient_Id)
```

3.4.3 Rewrite in terms of other views

A source tier can use views to provide redundant interfaces to its data. The same information might be available through a source table and through a view, and both may be available for queries. This means that there can be many ways to rewrite a query in equivalent form. In particular, many databases (e.g., warehouses) include materialized views, and query optimizers are beginning speed execution by rewriting queries to use them. These techniques can be used to find additional access permissions.

For a simple example of such query rewrites, consider query Q2 again. The source administrator could define a view containing the non-confidential information from PROCEDURE, as follows:

```
Create view PUB_PROCEDURE as
Select  Patient_Id, Doctor, Date
From PROCEDURE
```

This view can then be used to rewrite Q2 as:

```
Select Insurance_Co
From PATIENT
Where Patient_Id in (select Patient_Id from PUB_PROCEDURE)
```

The fact that this rewriting exists means that a user can execute Q2 if she has access permission on PATIENT and PUB_PROCEDURE. It may not mean, however, that Q2 will be executed this way. Once the system has established that the user has permission to execute the query, it may execute it any way it chooses. The chosen execution strategy may involve accessing tables (such

as PROCEDURE) that are not available to the user, but improve execution efficiency. The system can guarantee that the user will see only information she is allowed to see.

3.5 Physical Resource Permissions

Thus far, we have been concerned about permissions on information, applicable wherever the information resides. We anticipate that most enterprises will allow physical system owners to further restrict who may use their resources. Motivations include financial (only those who pay), workload management, and security (high security machines that contain copies of public information do not invite the public in).

For example, suppose that the CustomerService department of a company has gathered a great deal of customer information, and imposed appropriate access predicates to protect confidentiality. Basic information (name, address) is already widely used (e.g., by Shipping), but other departments want more. Top management has declared the information to be an enterprise resource (subject to confidentiality). But CustomerService argues that its server will be swamped, and it lacks hardware and personnel to run a larger operation.

The impasse is resolved when CustomerService restricts machine access to a role *Subscriber*, whose members have agreed to pay a fee for each access to this Customer information. Information permissions are checked separately (e.g., Shipping should not be allowed to see the customer's payment record). Credit-Decisions, which makes limited use of the data, pays a charge for each customer it examines. Marketing, which wants heavy usage, makes the marketing DBA a subscriber. She downloads weekly to the marketing server.

In order to support such scenarios, we allow administrators to specify *physical resource permissions* on stored tables. For example, multiple machines might maintain copies of the same data, for different user sets. Physical resource permissions determine whether the execution strategy may use a physical resource, i.e., a stored table or (in OO systems, a server).

The declaration *AllowPR(T,p)* specifies that physical resource permission p is enabled on stored table T (a source or a materialized view). The execution strategy must be expressed in terms of stored tables for which the user received such explicit allowances. Inference calculates whether some rewrite can provide sufficient physical resources. An administration tool may help relate the two kinds of permissions, e.g., to identify information permissions that have no physical resources to satisfy them. (It is useful to allow such orphans, rather than require information policy makers to deal immediately with physical resources.)

Physical resource permissions are inferred using rules analogous to those of Section 3.2, but with each permission set suffixed _PR. Thus each table T (stored or view) has two inferred sets of permissions. Its information permissions specify who is allowed to access T; its local resource permissions specify who will be physically able to calculate T, based on enabled access to underlying tables. A user can access a table only if she has both kinds of permissions on it.

3.6 Determining Enforcement Strategies

Federations, warehouses, and centralized DBMSs' views differ in where one tests permissions. Warehouse queries never reach the sources, so permissions *must* be checked at the warehouse. A centralized DBMS often tests permissions at compile time.

Federations have greater flexibility, assuming that all tiers know the requestor's identity.[10] Enforcement at the sources gives the data providers higher assurance because the federation need not be trusted. Enforcement at the derived tier tends to be more responsive and intelligible to users. Enforcing at both tiers would combine both advantages, and may be cheap (especially if done largely at compile time).

In an ideal world, these distinctions would not concern business decision-makers. Instead, the business decision would specify requirements for assurance and responsiveness, and a tool (or an always-available technical expert) would determine what access controls each tier enforced.

4 Conclusions and Open Questions / Research Issues

4.1 What We have Accomplished

Several important classes of systems (federations, warehouses, conceptual views) can be described as "multi-tier databases". Current metadata management tools offer little help in coordinating any but the most basic metadata (e.g., tables and their attributes). As a consequence, metadata is frequently absent or inconsistent, causing security vulnerabilities on one hand, and on the other, inadequate accessibility.

Our scenarios illustrated practical requirements for managing security metadata in such environments. We showed the need for communicating access permissions between different tiers in the database. We also illustrated the need for comparison capabilities, both for ordinary administration and for auditing. The scenarios further illustrated extra permissions granted when a view has filtered or summarized information.

We then sketched a technical foundation for coordinating access permissions. The key innovations that helped us simplify the model were to:

- Base the theory on query rewrite, rather than building a more complex theory of permission inference.
- Specify the global information permissions separately from local resource controls. Each has a strong inference rule, and the desired behavior is obtained as their intersection.

A series of examples showed how query rewrites permitted additional access. We then used the same theory to handle non-Read operations, and other granularities.

The theory in the paper is neither startling nor complex. Paradoxically, this is one of the key strengths of the paper. The security administration problem for warehouses and federations is broad and murky. We have identified a problem whose solution can give significant practical benefit, and can be implemented using relatively simple theory.

4.2 Technology Insertion Issues

It seems quite feasible to add metadata-propagation (e.g., for access permissions) to existing schema-management environments (for federations and warehouses). If we could automate coordination of, say, 50% of the metadata, we could make federation and warehouse administration cheaper and more effective. We believe that metadata propagation capabilities would be an excellent addition to a vendor's tool suite. Reference [Ros98b] shows how such a tool might look.

[10] Some products send all requests from "federation", a privileged subject.

Security administration seems a good place to begin metadata propagation tools. Security is mandatory. In contrast, it is harder to provide strong incentives for data quality and many other forms of metadata.

Perhaps the biggest technical challenge is to exploit rather than rebuild query rewrite technology, in generating propagation rules. First, query rewrite techniques require that derivations be understandable, e.g., be in SQL and definitely not in Java. Second, we want to be able to submit a query for rewrite but not execution.

4.3 Open Research Problems

The multi-tier paradigm opens many questions for security and other research. The issues in section 4.2 apply to many kinds of metadata. The discussion below focuses on access permissions.
First, we need a strategy for dealing with different permission granularities. Some organizations or DBMSs may permit multiple granularities simultaneously, while others will insist on a favored one.

Second, once the policy is known, administrators need help in devising an enforcement strategy. The software used at the different tiers may differ in flexibility, assurance, and granularity.

Third, we need a theory of permission deltas. Even a single table can involve an intimidating amount of metadata. We want the ability to propose a small change, and have just the small change be seen at other tiers. Also, when a view's users want more permissions than the view offers, we need algorithms that generate a minimal view on which one can assert the additional permissions.

Fourth, SQL's access controls on views are too inflexible to apply to federations. They require that each view definer be trusted by owners of source tables, and be willing to act as security administrator. And they don't allow the separate administration of local physical resources. As much as possible, we would like to see a federation or data warehousing system as an integrated database. Since SQL dominates relational databases, we want to integrate our approach with SQL security capabilities.

Fifth, we intend to investigate the possibility of security components. Enterprise Java Beans controls transaction behavior via property sheets; perhaps a similar approach would work here.

Sixth, the essence of our approach is to use some simple principles to propagate security metadata between the tiers. We believe that it would be fruitful to apply a similar approach to other forms of metadata. For non-security issues, the chief problems are to devise:

- theory for propagating other kinds of metadata (e.g., data pedigree, credibility, precision, timeliness), through a wide set of query operators (e.g., outerjoin, pivot);
- a component framework that allows administrators to incrementally insert and customize the ever-increasing set of propagation rules [Ros99a].

Finally, we have been concerned with semantics, not with algorithmic efficiency. Inference seems polynomial-time (in the size of the query rewrite space), but greater efficiency is desired, especially for avoiding full recomputation when a few permissions are added or revoked.

References

[Cas94] S. Castano et. al, *Database Security.* ACM Press Books, Addison-Wesley, 1994

[Inm96] W. Inmon, *Building the Data Warehouse.* John Wiley, 1996.

[Jaj95] S. Jajodia and C. Meadows, ``Inference problems in multilevel secure database management systems," in *Information Security: An Integrated Collection of Essays*, M. Abrams et al., eds., IEEE Computer Society Press (1995), pages 570--584.

[Kel86] A. Keller, "The Role of Semantics in Translating View Updates". *IEEE Computer*, Vol. 19, #1 (January 1986), pp. 63-73.

[Osz98] T. Oszu and P. Valduriez, *Principles of Distributed Database Systems.* Prentice Hall, 1998.

[Ros98] Arnon Rosenthal, Edward Sciore, "Propagating Integrity Information among Interrelated Databases", *IFIP 11.6 Workshop on Data Integrity and Control*, 1998. http://www.cs.bc.edu/~sciore/papers/IFIP98.doc

[Ros99a] Arnon Rosenthal, Edward Sciore, "First-Class Views: A Key to User-Centered Computing and Slimmer Tiers", submitted to *SIGMOD Record, 1999.* http://www.cs.bc.edu/~sciore/papers/fcviews.doc

[Ros99b] Arnon Rosenthal, Edward Sciore, Gary Gengo, "Demonstration of Multi-Tier Metadata Management" (html, gzipped) and "Warehouse Metadata Tools: Something More Is Needed" at http://www.cs.bc.edu/~sciore/papers/demo.zip and http://www.cs.bc.edu/~sciore/papers/mdtools.doc

[Sri96] J. Srivastava et al, "Answering Queries with Aggregation Using Views". VLDB, 1996, pp. 318-329.

[Sto75] M. Stonebraker, "Implementation of Integrity Constraints and Views by Query Modification", ACM SIGMOD Conference, 1975, pp. 65-78.

[Thu95] B. Thuraisingham, "Security Constraint Processing in Multilevel Secure Distributed Database Systems", *IEEE Transaction on Knowledge and Data Engineering,* Vol. 7, #2, April 1995.

Acknowledgement: Scott Renner has provided many insightful comments throughout this project. Gary Gengo and Tom Lee provided informed close readings of the text.

Multilevel Security Session

Wednesday, July 28, 1999

Chair: Pierangela Samarati
University of Milan

Enforcing Integrity While Maintaining Secrecy

Donald G. Marks
National Institute of Standards and Technology
100 Bureau Dr., Stop 893
Gaithersburg, Md. 20899-0893
e-mail - **donald.marks@nist.gov**

Abstract

We consider the role of constraints in maintaining both secrecy and integrity in a multilevel secure database. In a multilevel database, certain integrity and classification constraints create a secrecy problem since data additions, deletions or modifications require accessing data at higher levels. In many cases, however, these constraints may be approximated by a series of simpler constraints, called level-valid constraints. Level-valid constraints do not require access to any data that is classified higher than the data to be modified. Moreover, they meet the integrity requirements since any database state that satisfies the level-valid constraints also satisfies the multilevel constraints. Simple tests are developed to ensure the validity of proposed level-valid constraints and these constraints are derived for common cases of multilevel constraints.

1. Introduction

Consistency, or data integrity, in databases is usually achieved by associating with each database a set of *constraints*. The database management system (DBMS) has the responsibility to ensure that these constraints are satisfied by the database state at all times. Generally, this is done by checking the constraint for each tuple that is added, deleted or modified. A multilevel secure (MLS) DBMS has the additional responsibility of preventing improper disclosure[1] of information either by direct or indirect means. It is well known that there are inherent conflicts in MLS databases between the secrecy requirements and certain types of integrity constraints [DENN86]. In particular, it appears impossible to enforce certain integrity constraints without violating the secrecy requirements. Since the enforcement of multilevel constraints involves a trade-off between secrecy and integrity, the usual approach is to accept one or the other.

In this study, we show how it is often possible to maintain secrecy while still enforcing integrity. The approach taken will be to translate the original multilevel constraint, whose satisfaction requires the MLS DBMS to access both High and Low data, into a collection of *level-valid*[2] constraints. Each level-valid constraint has a fixed security level associated with it and is evaluated by referencing only data at or below that level. In a sense, then, we polyinstantiate the constraint rather than the tuple. That is,

[1] In this paper, we address only mandatory access controls

[2] From [QIAN94a].

instead of having one constraint that applies to all levels, each level will have its own constraint that only applies to that level.

More formally, suppose we have a database state, D and a multilevel constraint M. To enforce both the constraint and the security policy, we wish to replace M by a collection of level-valid constraints L, such that:

$L = \{L_1, L_2, ...L_n, M\}$ (where L_i and L_j are not necessarily unique)

1. Each L_j in L is a level-valid constraint at level j, and

2. If D satisfies level-valid constraint L_j, then D also satisfies L_{j+1}.

At the highest level, the constraint is still M, so there may exist database states that are allowable under the original multilevel constraint but not under the derived level-valid ones. Processes allowed to modify the database in a way that meets the original multilevel constraint, but not the appropriate level-valid constraint, will still have to be *trusted* at the higher level (note that this higher level constraint is not necessarily M). Thus, our approach is still a compromise between security and integrity. However, the use of such trusted processes is reduced by this scheme. Intuitively, if the level-valid constraint represents all the pertinent information available at the lower level, the scheme would minimize the use of trusted processes. Actually forming constraints that minimize these trusted processes is still an area of research, however.

1.1 Previous Work

Within a "multilevel secure" model, as in conventional databases, constraints are a common means of controlling not only integrity, but also access and inference control. Other researchers note the issue of conflict between the multilevel security and database integrity requirements in DBMS design. In particular, [DENN86, AKL87, MEAD88, BURN90, and MAIM91], have all addressed this issue. Thuraisingham and Ford [THUR95], Sandhu and Jajodia [SAND93], and Qian [QIAN94] have addressed polyinstantiation (i.e. multiple data instances) issues associated with integrity constraints. Gupta and Widom [GUPT93] propose a similar technique for local verification of constraints in distributed databases, although no security issues are addressed. While some of the theoretical and design issues for this approach were discussed in [JAJO95], this paper focuses upon the mechanics and the process of translating multi-level constraints into equivalent sets of level-valid constraints.

1.2 Example 1.

Suppose we have the relation:

EMP*(Name, Salary, Position):* containing the names, salaries, and positions of the employees in a law enforcement agency. There are three positions: *patrolman*, *investigator*, and *special agent*. The constraints are:

1) all *patrolman* salaries are less than any *investigator*'s salary;

2) all *investigator* salaries are less than any *special agent*'s salary.

The classification levels are: 1) *patrolman* records are classified confidential (*C*); 2) *investigator* records are classified secret (*S*); 3) *special agent* records are classified top secret (*TS*). Subjects at one level are not allowed to read information at a higher level, nor are they allowed to write at any other level.

If a new patrolman is hired, that record is automatically classified *C*, and inserted by a *C*-level subject. The database must decide if this is allowed. At the *C* level, the level-valid constraint is:

L_C: New *patrolman* salary must be less than or equal to the highest existing *patrolman* salary.

This constraint is sufficient to guarantee that a new tuple does not invalidate the original constraint.

The highest cleared category, *special agent*, has access to all lower classified information and needs no special constraint, so:

L_{TS}: New *special agent* salary must be higher than the highest existing *investigator* salary.

Finally, if a subject attempts to write a new *investigator* tuple, there are two checks that must be performed: the salary value must be higher than any *patrolman*, but lower than any *special agent*. This will require a conjunctive constraint to specify both conditions as follows:

L_S: New *investigator* salary must be less than the highest existing *investigator* salary, and greater than the highest paid *patrolman*.

This constraint is sufficient to guarantee that a new tuple satisfies both of the original multilevel constraints.

This example illustrates integrity constraints which are affected only by insert operations and are sufficient to enforce the original multi-level constraint. Note that constraints that are enforced during an insert operation may be different from the constraints that are enforced during a delete operation.

The next sections will formalize and generalize the approach to provide methods of forming these level-valid constraints as well as assurances of their correctness.

2. THE GENERAL SOLUTION

2.1 Notation

The first necessary step is to define a specific formalism for constraints. Constraints are generally formed by comparing characteristics (given by a function) of two sets of data (given by a database view). This can be formalized in the following definition.

Definition of Constraint: *a constraint is an expression of the form*:

$f_1(q_1(R)) \ \Theta \ f_2(q_2(R))$. *Which is true for any valid database state.*

Where: q_1, q_2 are database views on the relation R; f_1, f_2 are functions defined on the data revealed by the database views and Θ defines a partial order relating the values derived by f_1, f_2. To avoid the complexities of considering the database representation, we adopt the *universal relation* viewpoint, so that the relation *R* is equivalent to the entire database.

The functions are defined using mathematical notation, i.e. f(x)=y, where x is the argument of the function. Note that f_1, f_2 may be any user defined functions, including those implemented by means of triggers, procedures or object oriented methods. Functions however, are limited to acting upon the information included in the view defined as their argument. Similarily, Θ may be any partial order, including *functional dependence* from relational database theory. However, commercial database systems only allow for a few comparators so in this study we assume that f_1, f_2 yield either a numerical value or a set of string values. We also assume Θ is one of the standard comparators; ($>$; $<$; $\leq$; $\geq$; $=$) for numeric values, or the subset inclusion operator "$\subseteq$", or its inverse "$\supseteq$" for string values (or *sets* of numerics). In later discussions, Ω will denote "Θ or = ", so if $\Theta \equiv <$, then $\Omega \equiv \leq$.

For any multi-level database, some of the data may be viewed by a subject cleared to level L, namely that information classified no higher than level L. We will denote this restricted view by the subscript L. Therefore, R_L will denote that subset of R which is visible at level L.

The function and view based notation presented here is more intuitive than previously proposed notations and allows the database itself to evaluate the constraints. It maintains separation of important concepts (views, functions, and tests), allows for multiple aggregate functions, and is not restricted to using base relations or conjunctions of simple selects.

In our notation, the constraint "all *patrolman* salaries are less than any *investigator*'s salary" becomes:

MAX(**select** *Salary* from *EMP* **where** *Position* = '*patrolman*') $\leq$ MIN(**select** *Salary* from *EMP* **where** *position* = '*investigator*').

More complicated constraints may require the following extension:

Definition 2: A complex constraint is an expression of the form

IC_i AND IC_j

where either both IC_i and IC_j are simple constraints as defined in Definition 1, or IC_i is a simple constraint and IC_j is a complex constraint.

2.2 Sufficiency

The critical feature of this notation is the fact that it provides us with a way to *order* the relations using the comparator Θ. The fact that the relations are ordered may allow us to derive some *simple* tests for determining if a tuple may be added to (or deleted

from, or modified in) the database. Theorem 1 uses this ordering to establish our a testable condition.

Theorem 1 – Tuple Addition: Given a constraint, $f_1(q_1(R)) \Theta f_2(q_2(R))$, which is known to be satisfied by the current state of the database, and a tuple, t, satisfying the following two conditions:

(1) $f_1(q_1(R \cup t)) \Omega f_1(q_1(R))$ and

(2) $f_2(q_2(R)) \Omega f_2(q_2(R \cup t))$,

then the database will still satisfy the constraint after t is added.

[Theorem 1 may be modified to address *delete* and *modify* in a straightforward manner.]

Theorem 1 specifies two conditions, one for each expression in the constraint. We can therefore define two sets of valid tuples, one for each condition. Those tuples in both sets may be added to the relation and still satisfy the original constraint.

[For real-life complex databases, Theorem 1 may have to be implemented using "before and after images". The values of $f_1(q_1(R \cup t))$ and $f_2(q_2(R \cup t))$ may need to be calculated from the "after" image, that is, after t is inserted *and committed*, to the database. Otherwise there may be additional assertions, triggers, constraints and procedures that would change other data in the database view.]

Fortunately, in many cases of practical interest, a substantial number of tuples are in both sets. It is even common for a tuple to influence only one of the conditions in Theorem 1, while the other condition is satisfied by all tuples. For example, if $f_2(R)$ is equal to a constant, k, then it is true that $f_2(q_2(R)) = f_2(q_2(R \cup t)) = k$, regardless of what tuple t is added to the database. In such cases, tuples only influence one condition, so the conjunction does not present a serious problem.

Example 2. Consider the constraint: "all *patrolman* salaries are less than any *investigator*'s salary". From Theorem 1, it follows that we need to satisfy the two conditions:

1) MAX{**select** *Salary* **from** (EMP $\cup$ *t*) **where** *Position* = *patrolman*} $\leq$

MAX{**select** *Salary* **from** EMP **where** *Position* = *patrolman*} and

2) MIN{**select** *Salary* **from** EMP **where** *Position* = *investigator*} $\leq$

MIN{**select** *Salary* **from** (EMP $\cup$ *t*) **where** *Position* = *investigator*}.

Patrolman tuples with salaries *less* than or equal to the present maximum will not change the value of the first condition in (1). All these tuples will therefore satisfy (1). *All patrolman* tuples will also satisfy (2) since it is unchanged by their addition, regardless of their salary. The addition of an *patrolman* tuple therefore only requires evaluation of one, level-valid constraint:

I_C: "if *t*.position = *patrolman*, then *t.Salary* $\leq$ MAX{**select** *Salary* **from** EMP **where** *Position* = *patrolman*}.

Note that, if each tuple affects only one of the views $(q_1(R))_L$ or $(q_2(R))_L$, then:

$(q_1(R))_L \cap (q_2(R))_L = \varnothing$, **(Separation Condition)**

and Theorem 1 may be applied at level L. This is a very powerful and useful test. It provides a very simple method to determine when the "multilevel" constraint may be separated into distinct classification levels or compartments. Obviously, if the Separation Condition holds, each new tuple will affect, at most, one of the test conditions in Theorem 1. This affected condition is visible at level L, and is the only condition that needs to be evaluated in order to prove the continued validity of the multilevel constraint.

3. COMMON DBMS OPERATIONS

3.1 Automated Constraint Derivation

The next step is to automatically propose level-valid constraints which may represent sufficient conditions for the addition of (deletion from, modification to) a tuple in a relation. We do not claim to have evaluated every variation of such constraints, nor to find an optimal solution to the problem. We have, however, attempted to partition the problem and devise appropriate rules for each node in the taxonomy. Using the results will require assigning the constraint to the appropriate node and applying the corresponding rules.

To make it possible to automate the derivation of level-valid constraints, we will assume that the addition of a tuple does not affect $f_2(q_2(R))$. We will also assume that the addition or deletion of a tuple does not cause and "cascade" effects due to other triggers or procedures. The technique could be extended to allow for such circumstances, but the results would have to be framed in terms of the less intuitive before and after images. In the following procedures, we are required to compare views at differing classification levels. Since we must do this comparison at the lowest of these levels, some changes must be made to the view definition. In transmogrifying a high level view into a lower level view we may either create a bigger or a smaller view (disjoint views would have no value). Much of the following discussion formalizes this rather simple observation.

The problem is approached on a case-by-case basis. This approach is appropriate since there are only a limited number of database operations and comparators to be analyzed. In commercial databases, choices of functions are limited to the common aggregate functions of COUNT, MAX, MIN, and SUM. The choices for comparators are similarly limited to $<$, $>$, $\leq$, $\geq$, $=$, $\supseteq$, or $\subseteq$. Each combination of a database operation and comparator comprises a case.

3.2 Relationship of Components

Assume a multilevel constraint $M = f_1(q_1(R)) \Theta f_2(q_2(R))$, where the addition of a tuple at level L, t, changes the value of $f_1(q_1(R))$ but does not affect $f_2(q_2(R))$. We will derive a view $q_3(R_L)$ such that:

$f_1(q_1(R)) \Omega f_1(q_3(R_L))$ always holds, and define the level-valid constraint as:

$L = f_1(q_3(R_L)) \Omega f_2(q_2(R))_L$.

Then, by transitivity, if L holds, M holds. The only unknown is the view definition $q_3(R_L)$. Note that the query q_3 is submitted by a subject at level L and hence can only operate on R_L - the restriction of R to level L. The query q_2, however, is submitted by a High cleared subject and so $q_2(R)$ may contain attributes classified higher than L. In order to compare the two relations, $q_2(R)$ must be restricted to level L, (i.e. $(q_2(R))_L$). Even though $q_1(R)$ and $q_3(R_L)$ may have differing attributes, they could contain some common tuples. A tuple in $q_1(R)$ will match a tuple in $q_3(R_L)$ if the two tuples have identical primary keys and $(q_1(t))_L = q_3(t_L)$. We are especially interested in the situation where all tuples in one set are also in the other. Either $q_3(R_L)$ or $q_1(R)$ could be the larger set (called the superset) and have either additional tuples, or longer tuples. There are therefore two generic types of views of interest to us: the *superset* case where $q_1(R) \subseteq q_3(R_L)$ and the *subset* case where $q_3(R_L) \subseteq q_1(R)$.

Superset Views: $q_1(R) \subseteq q_3(R_L))$. All tuples in the multilevel view $q_1(R)$ are also in $q_3(R_L)$, and the level-valid view $q_3(R_L)$ may be larger.

These types of level-valid constraints are frequently formed by deleting those clauses restricting the view to high classified attributes. Thus the low-classified view "ID number of all SR71 airplanes" is certainly a larger set than the high-classified view "ID number of all SR71 airplanes on spy missions".

Subset Views: $q_3(R_L) \subseteq q_1(R)$. All tuples in the level-valid view $q_3(R_L)$ are also in the multilevel view $(q_1(R))$, and the multilevel view $q_1(R)$ may be larger.

Subset views are related to the aggregation problem where a subset of High information is classified Low either by design or because it is common knowledge. This might occur if the view "ID number, Mission, of all SR71 airplanes" is classified High, but the view "ID number of all SR71 airplanes" is available to flight crews and so must be classified Low. This can also occur if certain attributes are classified High, or if cover-stories are implemented. The Low-level view results in a subset of the High-level one.

3.3 Additions/Deletions

Fortunately, for the database functions being considered (COUNT, MAX, MIN, SUM), there is a direct relationship between the value of the function and the size of the set operated on by that function. That is, $MAX(A) \leq MAX(B)$ if $A \subseteq B$, and similarly for COUNT and SUM (of positive values). For the functions MIN, and SUM (of negative values) there is an inverse relationship, so $f(A) \leq f(B)$ if $B \subseteq A$. Therefore, given f_1 and Θ,

we can determine whether $q_3(R_L)$ should be a subset or superset view. Indeed there are only four combinations, or cases, to consider.

Case 1. Assume:

(1) f_1 is COUNT, MAX or SUM (for positive values of *t*);

(2) Θ is $<, \leq, =$, or $\subseteq$.

(3) $q_3(R_L)$ is a subset view.

If adding a new tuple increases $f_1(q_1(R))$, then $f_1(q_3(R_L))$ will not increase by a greater amount, so

$f_1(q_3(R_L)) \; \Omega \; f_1(q_1(R))$ always holds.

For example: COUNT(SELECT(ID-Number FROM DB WHERE Plane_Type=SR71 AND Mission_Classification < Confidential)) ≤ COUNT(SELECT(Name FROM DB WHERE plane_type=SR71)) will always hold, regardless of Mission_Classification.

Case 2. Assume:

(1) f_1 is MIN, or SUM (for negative values of *t*);

(2) Θ is $>, \geq, =$, or $\supseteq$.

(3) $q_3(R_L)$ is a subset view.

If adding a new tuple decreases $f_1(q_1(R))$, then $f_1(q_3(R_L))$ will not decrease by a greater amount, so

$f_1(q_3(R_L)) \; \Omega \; f_1(q_1(R))$ always holds.

For example: MIN(SELECT(Emp_Pay FROM (DB WHERE Job = analyst))) ≥ MIN(SELECT(Emp_Pay FROM DB WHERE Job = analyst OR Job = agent)) will always hold (assume "agent" records are classified higher than "analyst" records).

Case 3. Assume:

(1) f_1 is COUNT, MAX or SUM (for positive values of *t*);

(2) Θ is $>, \geq, =$, or $\supseteq$.

(3) $q_3(R_L)$ is a superset view.

Then an increase to $f_1(q_3(R_L))$ will not always increase $f_1(q_1(R))$, so:

$f_1(q_3(R_L)) \; \Omega \; f_1(q_1(R))$ always holds.

For example: COUNT(SELECT(Name FROM DB)) ≥ COUNT(SELECT(Name FROM DB WHERE Real_Agency .eq. "CIA")) will always hold, regardless of the value of Real_Agency.

Case 4. Assume:

(1) f_1 is MIN, or SUM (for negative values of *t*);

(2) Θ is $<, \leq, =$, or $\subseteq$.

(3) $q_3(R_L)$ is a superset view.

Then if adding a new tuple decreases $f_1(q_3(R_L))$ it will not always decrease $f_1(q_1(R))$, so:

$f_1(q_3(R_L))\ \Omega\ f_1(q_1(R))$ always holds.

For example: MIN(SELECT(Mission_Cost FROM DB)) ≤
MIN(SELECT(Mission_Cost FROM DB WHERE Mission .eq. "spy")) will always hold, regardless of the value of Mission.

For other combinations of q_3, f_1, and Θ (i.e. f_1 = COUNT, $\Theta \equiv \geq$, and $q_3(R_L) \subseteq q_1(R)_L$), there will exist a level-valid constraint which is sufficient to validate the multilevel constraint for tuple *deletions*. That is, for the complimentary conditions, tuples can be deleted (but not added) without checking the original multilevel constraint, although the multilevel constraint must still be checked if tuples are added to ensure that the new set is still small enough to satisfy the constraint. If f = identity, $f(q_1(R)) = q_1(R)$ so $(q_1(R))_L = q_3(R_L)$. In dealing with identity functions, we must establish both subset and superset conditions, so that $(q_1(R))_L = q_3(R_L)$.

3.4 Modify Operations

A "modify" operation cannot be handled as two separate operations but must be considered as an atomic transaction. The basic principles derived above must still apply, that is, the Low view will still be either a subset or a superset of the High view. However, it is no longer clear if the function f_1 is increasing or decreasing. In fact, this will normally depend upon the actual values for the attributes in the new and old copies of tuple *t*. For example, if f_1 = SUM, then f_1 is increasing when $|q_3(t)|>0$, and f_1 is decreasing when $|q_3(t)|<0$.

Table 1: Level valid tuple modification constraints

q_3 - *Superset* (i.e. $q_1(R) \subseteq q_3(R_L)$)			q_3 - *Subset* (i.e. $q_3(R_L) \subseteq q_1(R)$)		
f_1	Θ	conditions on t_o and t_n	f_1	Θ	conditions on t_o and t_n
COUNT	all	none	COUNT	all	none
MAX	≥, =	none	MAX	≥, =	$q_3(t_n)\ \Theta q_3(t_o))$
MAX	≤, =	$q_3(t_n)\ \Theta q_3(t_o))$	MAX	≤, =	none
SUM	≤, =	$q_3(t_o) - q_3(t_n) \geq 0$	SUM	≤, =	none
SUM	≥, =	none	SUM	≥, =	$q_3(t_o) - q_3(t_n)\ 0$
MIN	≥, =	$q_3(t_n))\ \Theta\ q_3(t_o)$	MIN	≥, =	none
MAX	≤, =	none	MAX	≤, =	$q_3(t_n)\ \Theta q_3(t_o))$

Table 1 presents these additional conditions which must be evaluated in order to determine the behavior of these functions. These conditions become simple tests for the validity of modifying tuples. In Table 1, $L = f_1(q_3(R_L - t_o \cup t_n)) \ \Theta \ f_2(q_2(R))_L$ is the constraint for modification of tuples $(t_o \rightarrow t_n)$ where the RHS of $f_1(q_1(R)) \ \Theta \ f_2(q_2(R))$ is unaffected by tuple modification.

3.5 Limitations

If any of the following are true we can make no statement concerning the existence of a level-valid constraint: (1) $q_3(R_L)$ is neither a subset nor a superset of $q_1(R)$; (2) f_1 is not monotonic with respect to set sizes; or (3) Θ does not specify a *partial order* relation. This does not mean that such level-valid constraints do not exist, merely that their existence cannot be proven by this mechanism. Usually heuristic analysis is required.

Fortunately, the situation, $f_1(q_3(R_L)) = f_1(q_1(R))_L$, is quite common and satisfies both *Superset Class* and *Subset Class* requirements. In such a case, level-valid constraints will exist for both addition and deletion of tuples.

3.6 Constraint View Derivation

We have two choices for views (subset or superset), two choices for functions (increasing or decreasing), and only two real choices for comparators (<, >). In practice, only the view in the level-valid constraint will differ from the one in the multilevel valid constraint. We will examine each choice in more detail and summarize the results as rules that may be applied to the multilevel view to derive the level-valid view. In the following discussion, we consider each view q(R) as being decomposed into the unary and binary functions available in a relational database. In this approach, the "q" consists only of "select/project" operations, while the "R" may represent a base relation or one formed by a Cartesian product, complement, or union.

3.6.1. The Superset View: $q_1(R) \subseteq q_3(R_L)$

To derive a level-valid superset, $q_3(R_L)$, of a multilevel view, $q_1(R)$, we must define a new level-valid view, operating only on data at level L or below, that still contains all the tuples originally in $q_1(R)$. The first thing to note is that, while $q_1(R)$ references High classified information, a level-valid view cannot actually return High classified information. Since we must keep all tuples that satisfy $q_1(R)$ and these must all be classified L or below, q_1 cannot contain "project" clauses referring to higher classified attributes. This leads to the first rule:

Superset Rule 1: For *Superset Class* situations, a level-valid constraint is not derivable if q_1 contains a *project* clause returning High classified attributes.

Generally, if q_1 does not contain High project operations, we can set $q_3 = q_1$, but eliminate *select* clauses referring to High attributes. Eliminating *select* clauses does not eliminate any tuples, the new query will simply have fewer restrictions and additional

tuples will satisfy it. This is why q_3 is regarded as specifying a superset of the original query, q_1.

Superset Rule 2: Eliminate *select* clauses relating to High attributes.

We now have two rules explaining how q_3 differs from q_1. We must also know if the relation, R, needs to be changed. The project and select operations apply to a database relation. There are three additional operations that may be used to define that relation: Cartesian product; union; and complementation.

(a) Cartesian Product: Consider $q_1(R)$ where $R = R_L x R_H$ with R_H being classified High. Tuples formed via Cartesian products will contain some Low classified attributes and some High classified attributes. However, Since q_3 contains no clauses, either *project* or *select*, referring to High classified attributes, the view $q_3(R_L x R_H)$ cannot contain any tuples with High classified attributes. The tuples satisfying the query therefore cannot contain more information than those tuples satisfying $q_3(R_L)$, and hence could be formed directly from R_L. It is possible that the Cartesian product could create duplicate tuples, but these would be eliminated in the view, so $q_3(R_L x R_H) \subseteq q_3(R_L)$.

(b) Union: Consider $q_1(R)$ where $R = R_L \cup R_H$. Suppose that $q_1(R_L \cup R_H) = q_1(R_L) \cup q_1(R_H)$, and $q_1(R_H) \neq \varnothing$. Then q_1 contains *project* clauses relating to High classified attributes, and we cannot generate a superset case.

(c) Complementation: Since R_L and R_H are disjoint, they have no tuples in common. Consider first $q_1(R_L - R_H)$. But $q_1(R_L - R_H) = q_1(R_L) - q_1(R_H)$, which we replace with: $q_3(R_L) - q_3(R_H)$. So $q_3(R_L)$ is larger than $q_1(R_L - R_H)$, and we have a superset case. Therefore, $R_L - R_H$ may be replaced with R_L to form a superset case. Consider now $q_1(R_H - R_L)$. This reduces to: $q_3(R_H) - q_3(R_L)$, but $q_3(R_H) = \varnothing$ (the null set), so "removing tuples" is not defined, and we must set $q_3(R_H - R_L) = \varnothing$. So, in cases other than High complementation, the High relation may simply be removed from consideration.

Superset Rule 3:

If $R = R_L \times R_H$, or $R = R_L - R_H$, set $R = R_L$.

If $R = R_H - R_L$, set $q_3(R_L) = \varnothing$.

Similar rules may be derived for subset views; $q_3(R_L) \subseteq q_1(R)$

3.6.2. Example 3:

Assume a common database consisting of two tables: EMP (employee, classified *L*) MGR (managers, classified *H*). The constraint "if an *employee*'s *manager* makes less than \$50k then the *employee* must make less than \$25k" (i.e. higher paid managers are allowed to hire higher paid subordinates). Assume that the *employee* record defines the salary, *Salary*, and the *employee*'s manager, *Mname*. Denote as *T* the "table" corresponding to the single employee tuple, *t*. The multilevel constraint for addition of an *employee* tuple then is:

M = (**Select** *t.Salary* **from** *T, MGR* **where** *T.Mname* = *MGR.Mname* **and** (*MGR.Salary*< $50K)) < $25k

Or, by separating out the natural join, this is equivalent to:

M = (**Select** *t.Salary* **from** (*T, MGR* **where** *T.Mname* = *MGR.Mname*) **where** (*MGR.Salary*< $50k) < $25k

Since we are dealing with an identity function, it may be considered either a subset or a superset case. We may eliminate the natural join and the High classified select clause, giving the level-valid constraint for an *employee* tuple as:

L = (**Select** *t.Salary* **from** *T*) < $25k

Note that this constraint also satisfies the tuple deletion conditions, so the database does not have to be checked for consistency if *employees* making less than $25k are either hired or fired.

Although the set of rules appears like a complicated and disconnected list of special cases and disconnected rules, in reality it is no more complicated than any other algebraic system. The process of generating a level-valid constraint is similar to other algebraic reduction procedures. Simply apply the right rule at the right time.

4. CONCLUSIONS

We have shown that many multilevel integrity constraints can be transformed into multiple level-valid constraints whose satisfaction is sufficient to ensure that the original multilevel constraint is also satisfied. The level-valid constraints are free from signaling channels.

This transformation is facilitated by devising a simple, powerful, yet intuitive method of expressing constraints as the relationship between the values of aggregate functions applied to views. In many cases it is possible to transform the original multilevel constraint into multiple level-valid constraints whose satisfaction is sufficient to ensure that the original is also satisfied. In transforming the multilevel constraint we have two possibilities: (1) the aggregate function remains valid if applied against a subset of the original tuples (i.e. the maximum value attained in a set will be at least as great as the maximum value attained in any subset) or (2) the aggregate function remains valid if applied against a superset of the original tuples (i.e. the maximum value attained in a set will be no greater than the maximum value attained in any superset).

Within this general solution, we can actually derive level-valid constraints for the common database operations and functions. We describe the procedures for generating the needed set of tuples as a view at level *L* or below, having the desired relationship (that is, being either a subset or a superset) to the view specified in the constraint. The generation of this new view is straightforward, but the techniques vary depending upon the relational algebra operations being used.

The techniques may be generalized for use with transactions, or constraints expressed as triggers, not simply the tuple additions/deletions considered already. These

new level-valid constraints do not solve the entire problem, and occasionally tuple modifications will have to rely upon trusted methods as are commonly used in current implementations. However, the situation is never more complicated, and level-valid constraints are usually simpler and more straightforward than the trusted process technique.

5. References

[AKL87] Akl, S.G. and D.E. Denning, "Checking Classification Constraints for Consistency and Completeness," *Proceedings, 1987 IEEE Symposium on Security and Privacy*, IEEE Computer Society Press, Los Alamitos, CA., 1987, pp.196-201.

[BURN90] Burns, R.K., "Referential Secrecy," *Proceedings, 1990 IEEE Symposium on Security and Privacy*, IEEE Computer Society Press, Los Alamitos, CA, May 1990, pp. 133-142.

[DENN86] Denning, D.E., T. Lunt, P. Neumann, R. Schell, *Secure Distributed Data Views – Security Policy and Interpretation for a Class A1 Multilevel Secure Relational Database System*, SRI International, Nov. 1986.

[GUPT93] Gupta, A., and J. Widom, "Local Verification of Global Integrity Constraints in Distributed Databases," *Proceedings of the 1993 ACM SIGMOD Int. Conf. on Management of Data*, 1993, ACM, pp.49-58.

[JAJO95] Jajodia, S., D. Marks, and E. Bertino, "Maintaining Secrecy and Integrity in Multilevel Databases: A Practical Approach," *Proc. 18th National Information Systems Security Conference*, Baltimore, Md., Oct. 1995, pp.37-49.

[MAIM91] Maimone, W.R., and R. Allen, "Methods for Resolving the Security vs. Integrity Conflict," *Proceedings, Fourth RADC Multilevel Database Security Workshop*, Little Compton, R.I., April, 1991, pp. 55-59.

[MEAD88] Meadows, C., and S. Jajodia, "Integrity Versus Security in Multilevel Secure Databases," in *Database Security, Status and Prospects*, Carl E. Landwehr, ed., North-Holland, Amsterdam, the Netherlands, 1988, pp.89-101.

[QIAN94] Qian, X., "Inference Channel-Free Integrity Constraints in Multilevel Relational Databases," *Proceedings, 1994 IEEE Symposium on Security and Privacy*, IEEE Computer Society Press, Los Alamitos, CA., May, 1994, pp. 158-167.

[SAND93] Sandhu, R.S., and S. Jajodia, "Referential Integrity in Multilevel Secure Databases," Proceedings, 16th National Computer Security Conference, Baltimore, MD., Oct. 1993, pp. 39-52.

The Effect of Confidentiality on the Structure of Databases

Adrian Spalka and Armin B. Cremers
Department of Computer Science III, University of Bonn
Roemerstrasse 164, D-53117 Bonn, Germany
Fax: +49-228-734 382, Email: adrian@cs.uni-bonn.de

Abstract

We give in this work an answer to the question of how real-life confidentiality can be introduced to an open database in a declarative manner. We present five forms of real-life confidentiality, translate them in the context of logic and show that only two forms can be brought in accord with databases that make the Closed World Assumption. These two forms, which correspond to real-life situations in which we do not want to admit that we are trying to keep something secret, can be precisely defined as distortions of the database's intended model. The main result of this work is a definition of a database with unequivocal static and dynamic semantics that supports real-life confidentiality at a declarative level. Two of its properties are that we never encounter polyinstantiation and that there is no inference.

1 Introduction

Many works on databases commence with the statement that a database is generally regarded as an image of a given section of the real world. Relational and logic-based databases are defined in such a way that their components correspond to many important elements of the real-world section.

Attribute values of a tuple or terms correspond to names that can be assigned to entities of the real-world section. A relation or a predicate symbol is used to express a simple statement on one or more entities' property or relationship. A tuple or a ground atomic formula is a concrete statement on the entities named in it. And the state of the database, ie a finite collection of tuples or ground atomic formulae, is a snapshot of the state of the properties. In a strict sense, the truth value of only those tuples which are contained in the state is known – the truth values of the remaining tuples are unknown. This situation, however, is not in accord with the situation in many real-world sections. To give an example, if a train at a particular time is not listed in the timetable, then everybody knows that there is no train at this time and if an item is not listed on the shipping order, then everybody knows that it is not shipped. To adapt the database to this real-life situation, Reiter (1984) introduced and formalised the Closed World Assumption (CWA), which tells us that a tuple is false if it is not in the database state. Integrity constraints, ie algebraic expressions or closed formulae, are also not a theoretical invention for its own sake but a reflection of existing invariants and conditions of the real-world section.

If we take a closer look at a database, we will notice that great care has been taken to define the above-mentioned components in such a way that, within the chosen expressive frame, they represent the best possible copy of the elements of the real-world section. We cannot say that this is also true if we take a look at confidentiality. The only means a database system offers is a blurred and inadequate concept of rights. It is blurred because the right to select an element and the rights to insert, delete and update elements are grouped together. But the first right relates to confidentiality, which is the ability to get to know something, and the remaining rights relate to competence and powers, which is the ability to do something. And it is inadequate because access restriction is merely an often necessary tool for achieving confidentiality but there are only a few situations in which it is also sufficient. In order to introduce confidentiality to a database as a sound and useful concept, we must first investigate its forms in the real life. We must determine what do we precisely mean and want to achieve when we say 'X should be kept secret from B.'; we must investigate the ways we use to achieve this and the as-

sumptions and preconditions on which the ways rely. And, lastly, we must find out which of the forms and ways can be expressed in and handled by a relational or logic-based database with the CWA.

Our approach is based on the following observation. The intended state of an open real-world section (OWS) is reflected in the intended model of an open database (ODB). A user who has access to the ODB can see all of it and modify it according to his powers. A database with confidentiality demands (CDB) must therefore correspond to a real-world section with confidentiality demands (CWS). The introduction of confidentiality into the OWS implies that the intended state of the OWS is distorted in front of a user from whom an element of the OWS should be kept secret – the OWS still exists, but the user is shown a CWS, which is a distorted OWS. To imitate this situation in databases, we must distort the intended model of the ODB and show the user a CDB such that the intended model of the CDB is a distortion of the intended model of the ODB.

We show in this paper that there are three possible kinds of distortions: those based on external conventions, single-model and multi-model, and that these distortions can be derived from the ways confidentiality is handled in the daily life. We also identify invariants of the ODB which cannot be distorted for if we do it the result can no longer be called a database. And, lastly, we present some criteria for the satisfiability of confidentiality demands.

The first kind of distortions stems from the observation that if somebody is looking at something he does not know, instead of telling him what it really is we can tell him that it is something else. This kind does not require any changes to the database for we only distort the external convention of the meaning of some database elements. To give an example, suppose there is a relation with the name hot_fields the intended conventional meaning of which is secret airfields. If a user is unaware of this convention and we want to keep it secret from him, we can tell him it is a list of favourite holiday spots.

Single-model distortions transform the ODB with its single model into a CDB with also only a single model. In real life, this process corresponds to situations in which we do not want to admit that we are trying to keep something secret. We present the user of the CDB with a single distorted model hoping that he is unable to notice the distorted parts of it. Suppose you want to go to a pub this evening and the person who asks you about your plans today should not know of it, so, you can try to put an innocent look on your face and tell him that you go to the library. This kind of distortions are particularly difficult to realise since you must ensure that all invariants are satisfied and the users' powers are not violated.

Lastly, multi-model distortions transform the ODB with its single model into a CDB with many models. The parallel to it in real life is a situation in which you admit that you have a secret or in which you pretend that you really do not know the answer to a question, though you do. The problem with this kind is that, by definition, a database with the Closed World Assumption always has a single model*. We also investigate the informal and formal sides of these distortions but we regard them as incompatible with our database.

In today's databases the static and dynamic semantics and integrity constraints have a clean declarative definition, whereas confidentiality is most often reduced to low-level operational access rules. This situation makes it hard, if not impossible, to conduct formal examinations and derive properties capable of proof with respect to confidentiality. The main contribution of this work is a declarative definition of confidentiality in databases with the CWA. The definition relies on the notion of a model which is a precise and well-understood element of mathematical logic. We also show that this is not a superficial definition – it reflects exactly two forms of confidentiality of the real life and, therefore, it is useful and easy to understand. It enables us to state confidentiality demands in the same declarative manner as, eg, we translate invariants of the real-world section into integrity constraints and we can formally investigate their satisfiability. The approach is undeniably more complex than a couple of access rules,

* If the application of the CWA to a database renders it inconsistent or yields several models, then the CWA is regarded is incompatible to the database and is not applied to it.

yet our secrets in the daily, real life prove that we are well able to handle it.

The subsequent section presents a formal definition of a database and an extensive example that illustrates the formal notions. Section three focuses on previous works in this area. In section four we introduce some extensions to an open database which are necessary preliminaries to the introduction of confidentiality. The central part of this work, section five, discusses the effect of the various forms of confidentiality on the structure of databases. In the last section a conclusion briefly summarises our findings and presents a short outlook.

2 Basic definitions

In this section we give a formal definition of an open database and illustrate it with a comprehensive example.

An alphabet $\mathfrak{A}$ is a non-empty and finite set of symbols, such that any string of $\mathfrak{A}$'s elements has a unique decomposition into $\mathfrak{A}$'s elements. $\mathfrak{A}^*$ denotes the set of strings of finite length of elements of $\mathfrak{A}$. Let $F = \mathfrak{A}^*$ be the set of function symbols, $P \subseteq \mathfrak{A}^*$ a finite set of predicate symbols, and $\rho_F : F \to \{0\}$ and $\rho : P \to \mathrm{N}_0$ the functions determining the arity of the symbols. Then

$$\Sigma = (\mathfrak{A}, P, \rho)$$

is a database signature. Let V be an infinite set of variables. Then $T(\Sigma) = F \cup V$ is the set of terms over the signature Σ. $T_G(\Sigma) = F$ is the subset of ground terms. The set of atomic formulae over Σ is

$$A(\Sigma) = \{p(t_1, \ldots, t_n) \mid p \in P, \rho(p) = n, t_i \in T(\Sigma), i = 1, \ldots, n\}$$

and $A_G(\Sigma)$ is the subset of ground atomic formulae. With $\alpha \in A(\Sigma)$, α is a positive literal and $\neg\alpha$ a negative literal. The first order language over the signature Σ is the smallest set $L(\Sigma)$ with the following properties: $A(\Sigma) \subseteq L(\Sigma)$; if $\varphi, \psi \in L(\Sigma)$, then $(\varphi) \vee (\psi) \in L(\Sigma)$; if $\varphi \in L(\Sigma)$, then $\neg(\varphi) \in L(\Sigma)$; if $\varphi \in L(\Sigma)$ and $X \in V$, then $\forall X : \varphi \in L(\Sigma)$. $L_0(\Sigma)$ is the subset of closed formulae, viz, all variables of a $\varphi \in L_0(\Sigma)$ are quantified. A normal clause is a closed formula $\alpha \leftarrow \lambda_1 \wedge \ldots \wedge \lambda_n$ in which all variables are assumed to be universally quantified, $\alpha \in A(\Sigma)$ is the clause's head and $\lambda_1 \wedge \ldots \wedge \lambda_n$, a conjunction of literals, its body. A clause is range-restricted if any variable that occurs in the clause occurs also in a positive literal in its body. $C_R(\Sigma) \subseteq L_0(\Sigma)$ is the set of range-restricted clauses over Σ.

Let $\mathfrak{D}$ and Ω be sets such that there is a $\omega_F \in \Omega$

$$\omega_F : F \to \mathfrak{D}$$

and for each $p \in P$, $\rho(p) = n$, there is a $\omega_p \in \Omega$

$$\omega_p : \mathfrak{D}^n \to \{\text{True, False}\}$$

Then $\hat{M}(\Sigma) = (\mathfrak{D}, \Omega)$ is an interpretation for the signature Σ. Alternative notations for restrictions to $\hat{M}(\Sigma) = (\mathfrak{D}, \Omega)$:

- for ground atomic formulae:
$$M(\Sigma) : A_G(\Sigma) \to \{\text{True, False}\}$$
- for closed formulae:
$$M_0(\Sigma) : L_0(\Sigma) \to \{\text{True, False}\}.$$

$\hat{M}(\Sigma)$ is a model of Φ, $\hat{M}(\Sigma) \in \mathrm{Mod}(\Phi)$, $\Phi \subseteq L_0(\Sigma)$, if $\forall\varphi \in \Phi : M_0(\Sigma)(\varphi) = \text{True}$. (Some works on databases define a model as

$$M^{-1}(\Sigma)(\text{True}) \subseteq A_G(\Sigma).)$$

$\hat{M}(\Sigma) = (\mathfrak{D}, \Omega)$ is a Herbrand-interpretation or a Herbrand-model if $\mathfrak{D} = T_G(\Sigma)$ and $\omega_F = \mathrm{id}$. $\Psi \subseteq L_0(\Sigma)$ is a logical consequence of $\Phi \subseteq L_0(\Sigma)$, $\Phi \models \Psi$, if $\mathrm{Mod}(\Phi) \subseteq \mathrm{Mod}(\Psi)$.

We assume that the intended semantics of a set $\Phi \subseteq L_0(\Sigma)$ is defined by its completion* with respect to Σ, $\mathrm{comp}(\Phi, \Sigma)$.

Let Σ be a database signature, $C \subseteq L_0(\Sigma)$, $\mathrm{Mod}(C) \neq \varnothing$, a finite set of closed formulae over Σ and $I \subseteq C_R(\Sigma)$ a finite set of range-restricted clauses over Σ such that $\mathrm{comp}(I, \Sigma) \models C$.† Then

$$D = (\Sigma, C, I)$$

is a logic-based database with completion semantics. Σ defines the database language, C is the set of integrity constraints, I is a valid present state and the intended semantics of I is defined by the unique Herbrand-model $M(\Sigma)$ of $\mathrm{comp}(I, \Sigma)$.

A transaction T is a finite sequence of finite sets of ground atomic formulae, which are annotated for insertion or deletion, ie

$$T = ((A_1, \vartheta_1), \ldots, (A_n, \vartheta_n))$$

---

* Cf, eg, Das (1992) or Cremers/Griefahn/Hinze (1994).

† This definition ensures that the integrity constraints are satisfiable and that the completion has a unique Herbrand-model.

such that $A_i \subseteq A_G(\Sigma)$ and $\vartheta_i \in \{\text{INS}, \text{DEL}\}$ for all $1 \le i \le n$. The operational semantics of T, ie the application of T to I is the successive insertion or deletion of T's sets, which yields a set I'.

$$I_0 = I$$
$$I_{k+1} = \begin{cases} I_k \cup A_{k+1}, & \text{if } \vartheta_{k+1} = \text{INS} \\ I_k \backslash A_{k+1}, & \text{if } \vartheta_{k+1} = \text{DEL} \end{cases}$$
$$I' = I_n$$

If I' is valid, then T is accepted and said to be valid and I' becomes the present state of D; otherwise T is rejected and the state remains unaltered. T is a singleton-transaction if $T = ((A, \vartheta))$ and $|A| = 1$. Let T be valid and $M'(\Sigma)$ the model of I'. The declarative semantics of T, ie the effect of T on $M(\Sigma)$ is a partitioned set $\tau = \tau_\delta \cup \tau_\iota$, $\tau \subseteq A_G(\Sigma)$, such that:

- $M(\Sigma)(\alpha) = \text{True}$ and $M'(\Sigma)(\alpha) = \text{False}$, for all $\alpha \in \tau_\delta$,
- $M(\Sigma)(\alpha) = \text{False}$ and $M'(\Sigma)(\alpha) = \text{True}$, for all $\alpha \in \tau_\iota$, and
- $M(\Sigma)(\alpha) = M'(\Sigma)(\alpha)$ for all $\alpha \in A_G(\Sigma)\backslash\tau$.

We now illustrate these definitions with a relational example.

EXAMPLE 1 Suppose we have a database with the following three relations:

Starships	Starship
	Enterprise
	Voyager

Missions	Starship	Destination	Objective
	Enterprise	Asgard	Fun
	Voyager	Midgard	Aid

Freight	Starship	Cargo
	Enterprise	Beer
	Voyager	Rice
	Voyager	Corn

The primary key of the Missions relation is its first attribute and both attributes of the Freight relation constitute its primary key.* There are foreign keys from Missions to Starships and from Freight to Starships on the Starship attribute.

Then:

- $\mathfrak{A}$ is a subset of the printable ASCII symbols and F the set of strings of finite length
- $P = \{\text{Starships}, \text{Missions}, \text{Freight}\}$, $\rho(\text{Starships}) = 1, \rho(\text{Missions}) = 3$, $\rho(\text{Freight}) = 2$
- $\alpha = \text{Freight}(\text{Voyager}, \text{Beer})$ is a ground atomic formula and also a range-restricted clause
- $\mathfrak{D} = T_G(\Sigma) = F$, and $\Omega = \{\omega_F, \omega_{\text{Starships}}, \omega_{\text{Missions}}, \omega_{\text{Freight}}\}$ with

$$\omega_F = \text{id}$$

$$\omega_{\text{Starships}} : \mathfrak{D} \to \{\text{True}, \text{False}\}$$
$$x \mapsto \text{False}$$

$$\omega_{\text{Missions}} : \mathfrak{D}^3 \to \{\text{True}, \text{False}\}$$
$$(x, y, z) \mapsto \text{True}$$

$$\omega_{\text{Freight}} : \mathfrak{D}^2 \to \{\text{True}, \text{False}\}$$
$$(x, y) \mapsto \begin{cases} \text{True, if } (x, y) = (\text{Voyager, Rice}) \text{ or} \\ \qquad (x, y) = (\text{Voyager, Gold}) \\ \text{False, otherwise} \end{cases}$$

define a Herbrand-interpretation† for $\Sigma = (\mathfrak{A}, P, \rho)$, the signature of the database.

- The database state is the set
  $I = \{\text{Starships}(\text{Enterprise}),$
  $\text{Starships}(\text{Voyager}),$
  $\text{Missions}(\text{Enterprise, Asgard, Fun}),$
  $\text{Missions}(\text{Voyager, Midgard, Aid}),$
  $\text{Freight}(\text{Enterprise, Beer}),$
  $\text{Freight}(\text{Voyager, Rice}),$
  $\text{Freight}(\text{Voyager, Corn})\}$
- The intended model
  $M(\Sigma) : A(\Sigma) \to \{\text{True}, \text{False}\}$
  of the state I assigns the truth value True to all elements of I and False to all other tuples.

* Note that the primary key of the Starships relation is implicitly given for it has only one attribute and I is a set, ie there are no duplicates.

† But note that it is not a model of the state.

- T = (({Freight(Enterprise, Coke)}, INS)) is an accepted singleton-transaction; its effect is $\tau = \tau_\delta \cup \tau_\iota$ with $\tau_\delta = \varnothing$ and τ_ι = {Freight(Enterprise, Coke)}. The previous model has assigned the truth value False to Freight(Enterprise, Coke), the present one assigns to it True. ⩓

3 Previous and related work

In SQL only the GRANT/REVOKE SELECT statements, which apply only to relations, support confidentiality. Formally, the open database $D = (\Sigma, C, I)$ with $\Sigma = (\mathfrak{A}, P, \rho)$ has a set of relations, P. If a user u does not possess the SELECT right to some relations, then there is a database $D_u = (\Sigma_u, C_u, I_u)$ with $\Sigma_u = (\mathfrak{A}, P_u, \rho_u)$ such that $P_u \subseteq P$ and the remaining components of D_u are restrictions of the components of D to the elements of P_u. Informally, we use this method to simply lie to the user: if an excluded relation is a base relation, then we lie to him about the extent of the real-world section which is known to the database and if the excluded relation is a view with an obvious connection to a base relation, then we lie to him about the truth value of the withheld tuples (they are true in the intended model of I but false in that of I_u). Note that D and D_u are not two independent databases. D_u is a distorted copy of D, which corresponds to a distorted image of the real-world section and any change applied to D_u is immediately propagated to D. The SQL-approach has two disadvantages. Firstly, there is no way of storing in D_u tuples that are not in D, ie, we cannot set a tuple's truth value to False in D and to True in D_u. And, secondly, we have not found any paper or manual, which tells you how to define P_u in such a way that the restriction of D to D_u results in a valid database D_u and that a transaction accepted by D_u will not be rejected by D, ie, that D_u retains the static and dynamic semantics of a database.

Most other algebraic approaches are based on SeaView, a multi-level relational data model introduced by Denning et al (1987). It assumes that there is an open database D which is associated with the highest security level of a lattice of security levels and that a database D_l is associated with any other security level l. Since each D_l is a restriction of D, the approach is in fact similar to that of SQL. The only difference is that in addition to reducing P, the set of relations, SeaView is also able to reduce I. Thus, SeaView simply lies to the user in the same way as SQL does. But whereas SQL provides a clear informal concept and formal definition of its notion of confidentiality, SeaView does not. The idea to bring NULL-values in connection with confidentiality and the idea that polyinstantiation – a phenomenon invented to justify a flawed design – is natural and inevitable resulted in situations in which the meaning of confidentiality is unclear. The obvious consequence of these facts is that SeaView also inherits all disadvantages of SQL and adds to it a pile of its own problems.

There are several works, eg Sicherman/de Jonge/van de Riet (1983), Morgenstern (1987) and Bonatti/Kraus/Subrahmanian (1992), which regard a formula as a unit of protection and a formula's confidentiality is defined as non-derivability. Whereas derivability is a unique notion, non-derivability can be formalised in three ways. Suppose that a formula $\varphi \in L(\Sigma)$ should be kept secret from the user u. Then, firstly, φ is not derivable in D_u if φ cannot be expressed in the language of D_u, ie, if $\varphi \notin L(\Sigma_u)$. Or, secondly, if φ is in D, then it is not in D_u or vice versa, ie, if the truth value of φ in D_u is opposite to that in D, ie $M(\Sigma)(\varphi) = \neg M_u(\Sigma_u)(\varphi)$. Or, lastly, if all true statements, which comprise φ, can be only reduced to a non-singleton OR-statement, ie, to a formula $\varphi \vee \psi$ with $\varphi \neq \psi$. This situation implies that D_u does not have a unique intended model but many and there are two models M_u^i and M_u^j such that $M_u^i(\Sigma_u)(\varphi) = \neg M_u^j(\Sigma_u)(\varphi)$. This last interpretation is chosen by Sicherman/de Jonge/van de Riet (1983) and Bonatti/Kraus/Subrahmanian (1992); Morgenstern (1987) does not tackle the choices.

Based on the conviction that standard logic is inadequate, Thuraisingham (1991) and Thuraisingham (1992) attempt to formalise the rules and properties of mandatory access control models* in NTML, a non-monotonic logic, which, however, has

* Cf, eg, Landwehr (1981).

been shown to be not sound. The main step in NTML is the assignment of a security level to all elements of the signature and to all formulae of D. $\Sigma_u = (\mathfrak{A}_u, P_u, \rho_u)$ is determined by the user's current security level and the components of D_u are restrictions of the components of D to the elements of $\Sigma_u = (\mathfrak{A}_u, P_u, \rho_u)$.

4 Users and powers in open databases

Before we can investigate the forms of confidentiality, we must realise that the ability to keep a unit of protection secret from a user is limited by two fundamental factors:

- the user's powers to change the present state, which must be respected
- the user's ability to learn about the unit's current state or existence without asking the database

Therefore, the first extension to an open database with respect to confidentiality is the inclusion of these concepts.

Let $D = (\Sigma, C, I)$ be an open database. We assume that:

- there is a finite set $U = \{u_1, ..., u_n\}$ of users who have legitimate access to D
- for each $u \in U$ there is a set $R_u \subseteq A_G(\Sigma)$ that reflects the user's powers in the real-world section, which means that a transaction T with the effect τ executed by u
  - is rejected if τ is not a subset of R_u, and
  - must be accepted by D if τ is a subset of R_u and does not violate the integrity constraints

  We say that T is authorised if τ is a subset of R_u.
- for each $u \in U$ there is a set $K_u \subseteq A_G(\Sigma)$ that reflects the user's specific knowledge on the real-world section, ie his ability to learn about the current state or existence of some units without asking the database

Then

$$D = (\Sigma, C, I, U, \{R_u\}_{u \in U}, \{K_u\}_{u \in U}) \qquad (1)$$

is an open extended database.*

The rights R_u determine the transactions user u is authorised to execute. While the decision on a transaction's validity is made by the integrity constraints, an organisation's or an application's security policy is solely responsible for the granting and the revocation of an authorisation. This means that we must distinguish the reasons because of which a transaction can be rejected. If it is because of a violation of the state's validity, then the integrity constraints make the reasons explicit, and, at the same time, provide the justification – both are visible and comprehensible to the user. If it is for lack of authorisation, then the user can neither find the reasons nor the justification within the database. The security policy is only reflected in the rights – it is not stated there. Thus if a user is puzzled about a missing right, then it is not the database, but the persons who set up the security policy that are responsible for providing a satisfactory explanation. Since this work is on confidentiality, the consequences of failing to do so can be neglected from our viewpoint as long as no information must be kept secret from the user. Otherwise, this behaviour may well affect a user's trustworthiness in the negative.

The purpose of the set K_u is that its elements cannot be kept secret from u. The question of how to determine K_u precisely has, of course, no single true answer. But the semantics of the powers, R_u, tells us that the least a user can know about the real-world section is what he has arranged or is able to do so, ie, R_u is a lower bound on K_u.

ASSUMPTION 1 $R_u \subseteq K_u$ always holds.

5 Confidentiality in databases

This chapter investigates the transition from an open database (ODB), which corresponds to an open real-world section, to a database with confidentiality demands (CDB), which corresponds to a real-world section with confidentiality demands.

5.1 Distortion of the intended external convention

The algebraic and logic-based theory of databases is not concerned with the question of how does a

* Note that there are no confidentiality demands, ie, the users are still allowed to know everything.

database user know the real-life meaning of the relation names and the attribute values, ie the elements of the signature. As long as the database is open, this question is, admittedly, of little avail to the theories. However, the situation changes as soon as we introduce confidentiality.

The actual correspondence between the elements of the real-world section and the elements of the signature is established in two steps. Firstly, the persons who communicate with the real-world section are identified with the users who communicate with the database. Secondly, the real-world meaning of an element of the signature is fixed in the intended external convention, a convention intelligible and known to all legitimate users.

In open databases the real-world objects and their properties are taken to be indistinguishable from their names in the signature of which the users have an implicit understanding. One could object that if this level of abstraction was questioned, then one could proceed ad infinitum, and the ironic end would be the situation in which one is not able to refer to a real-world object in a conversation but to point with a finger to this object itself. In practice, however, this argument misses the nature of codes, watchwords and other secret arrangements. Their usefulness relies on the fact that a group of people is in the know of only a single additional level of indirection with respect to some names' natural-language semantics. The sentence 'Tonight we go to the pub' gains a different meaning for the members of a gang who, in order to hide their intentions, have agreed to denote a bank as a pub. To take this situation into account, we assume that a signature is a syntactical object with semantic intentions in respect of its real-world meaning, which are stated in an explicit convention.

Let $\mathfrak{C}$ be the intended external convention of the signature* $\Sigma = (\mathfrak{A}, P, \rho)$. If an element of Σ should be kept secret from a user u, we can present to him $\mathfrak{C}_u$, a distortion of $\mathfrak{C}$.

EXAMPLE 2 Suppose that

- hot_fields is a relation, ie an element of P, and its intended external convention is $\mathfrak{C}$(hot_fields) = 'is a secret airfield'
- red $\in T_G(\Sigma)$ is an attribute value and $\mathfrak{C}$(red) = 'alert condition red'

If a user is unaware of the intended meaning of these symbols and we would like to keep it secret from him, then we can offer him the following external convention $\mathfrak{C}_u$, which is a distortion of $\mathfrak{C}$:

- $\mathfrak{C}_u$ (hot_fields) = 'is a favourite holiday spot'
- $\mathfrak{C}_u$ (red) = 'the colour red'

First of all, we do not gain any relevant insight if we try to formalise $\mathfrak{C}$ since it is the final level of describing the meaning of an image of a real-world section. For that reason the database has no way of influencing the intended external convention. On the one hand, this makes this distortion quite simple to apply because the database does not need to be changed in any way. On the other, its applicability is limited to only a few situations. Assumption 1 tells us that the user has no update rights to tuples with a distorted convention. Otherwise, we were in for a home-made sabotage. Suppose that user u of Example 2 changes red to green because he wants to give something a fresh look – this is a, possibly unintended, distortion of the open database with respect to the intended external convention and it confuses all users who rely on it. It is also useless in many other situations. It may be only of some help if the user is a guest who somehow sees a part of the database.

Since distortions of the intended external convention cannot be influenced by the database we do not examine it further and assume from now on that all users $u \in U$ have a uniform intended external convention.

5.2 Multi-model distortions

Multi-model distortions are a formalisation of two situations:

i) somebody pretends not to know the answer to a question though he does

ii) somebody admits to have a secret

The intention in both cases is to construct a CWS by introducing a certain degree of blur into the OWS.

An example of the first case is the question

* Note that a convention does not apply to the tuples' truth values. All elements of the database state are taken as true and all remaining ones as false.

'Where does this plane go to?' and the answer to it 'I really do not know'. Since you know or see that the plane has left, this answer tells you that the plane may go to any airport within its reach and on which it can land. Suppose that the meaning of the predicate W is 'is the destination of this plane', then, formally, this answer can be represented as the following finite disjunction:

$W(\text{Rome}) \vee W(\text{Paris}) \vee \ldots \vee W(\text{London})$

if Rome, Paris and London qualify as possible destinations. We can describe this situation with respect to the open database with its unique intended model $M(\Sigma)$ as follows. Suppose that Rome is the intended destination. Then the CDB for the user u, who asks the question, is constructed by replacing $M(\Sigma)$ with a finite set of models $\mathfrak{M}_u(\Sigma)$ such that:

i) The CDB claims that $\mathfrak{M}_u(\Sigma)$ is the smallest set of its models

ii) $|\mathfrak{M}_u(\Sigma)| > 1$, ie the state of some elements of the real-world section is unknown and there is no unique intended model

iii) $M(\Sigma) \in \mathfrak{M}_u(\Sigma)$, ie the intended destination is among the possible destinations

iv) There is at least one $M^i_u(\Sigma) \in \mathfrak{M}_u(\Sigma)$ such that $M^i_u(\Sigma)(W(\text{Rome})) = \text{False}$, ie at least one choice tells you that Rome is not the intended destination

An example of the second case is the question 'How much do you earn?' and the answer to it 'I know it but I will not tell it'. Informally, you are told that the amount of the salary is secret. Formally, the answer can still be represented as a finite disjunction since the lowest amount is fixed by law and the highest amount by common sense. The formal representation of this case in the database is identical to that of the first case with the exception of points (i) and (ii):

i) There is an ODB with a unique intended model and the CDB is a distortion thereof

ii) $|\mathfrak{M}_u(\Sigma)| > 1$, ie the state of some elements of the real-world section is deliberately blurred

Of these two cases the first one is not applicable to our database because it violates its definition. The statement that the OWS cannot be represented more precisely than by a non-singleton set of models, $\mathfrak{M}_u(\Sigma)$, contradicts the CWA, which, if compatible to the database, always yields a unique intended model.

The second case has several problems. Firstly, in order to give an indefinite answer, the semantics of the database has to be extended. Secondly, there are no satisfactory approaches to the representation of $\mathfrak{M}_u(\Sigma)$ and its operational treatment. Whereas these problems can be solved, its main flaw is shown by Sicherman/de Jonge/van de Riet (1983). They use a censor who takes into account the user-knowledge and the dialogue up to that time and who decides on the database's answer. The censor, who never lies, gives the indefinite answer if he believes that the user will be able to deduct the secret with the help of a precise yes/no-answer. The authors demonstrate that this can still be of little avail since the user can make use of the censor's reasoning in his own one and discover the secret.

Thus, we conclude, multi-model distortions cannot be used to introduce confidentiality to an open database.

5.3 Single-model distortions

Let us now turn our attention to situations in which confidentiality is understood in the sense that we would also like to keep the fact confidential that we have a secret.

5.3.1 Distortion of the truth values

An example of such a situation is the question 'Where does this plane go to?' and the answer to it 'To London', though you know that Rome is its intended destination. Since you also want to keep it secret that your answer is a lie you must take many additional precautions. Firstly, you must ensure that your answer complies with the general knowledge on the real-world section, eg, that this plane can indeed fly to and land in London. Secondly, you must ensure that the person who asks the question is unable to modify your secret and any information related to it, eg, that he cannot order this plane to Paris or add a cargo of grapes, which never arrives in London. And, lastly, when confidentiality is no longer desired you must inform this person of its true destination, eg Rome, and remove the lie used to cover it, eg London.

This situation has the following formal correspondence. Suppose that the user u asks the question. Then you present to u a CWS with the following properties:

- You pretend in front of u that the statement 'This plane flies to Rome' has the opposite truth value in the CWS as in the OWS
- In order to satisfy the integrity constraint 'Every plane has a destination' you use an alias, London, and pretend in front of u that the statement 'This plane flies to London' has the opposite truth value in the CWS as in the OWS
- You must check that u does not have the powers to alter the plane's destination and cargo
- You must record that the statement 'This plane flies to Rome' should be kept secret in front of u and that the statement 'This plane flies to London' is only used as an alias for the first statement, so that when the truth value of the first statement in the CWS is adjusted to that in the OWS, the same must be done for the alias.

With this in mind, it is easy to translate the situation in the context of our database. Recall that the intended model of the ODB

$$D = (\Sigma, C, I, U, \{R_u\}_{u \in U}, \{K_u\}_{u \in U})$$

is the mapping

$$M(\Sigma) : A_G(\Sigma) \to \{\text{True, False}\} \qquad (2)$$

which assigns a truth value to all ground atomic formulae, ie to all possible tuples. We now construct the CDB $D_u = (\Sigma_u, C_u, I_u)$ with

$$M_u(\Sigma_u) : A_G(\Sigma_u) \to \{\text{True, False}\}$$

Let α be the tuple which should be kept secret. Then:

i) set

$$M_u(\Sigma_u)(\alpha) = \neg M(\Sigma)(\alpha)$$

ie set the truth value of α in the CDB to the opposite of its truth value in the ODB

ii) if there are some integrity constraints $\psi_1, \ldots, \psi_m \in C_u$ which are no longer satisfied due to step (i), try to find one or more aliases $\beta_1, \ldots, \beta_n$ such that $\psi_1, \ldots, \psi_m$ are satisfied if we set

$$M_u(\Sigma_u)(\beta_i) = \neg M(\Sigma)(\beta_i)$$

for all $i = 1, \ldots, n$

iii) check that $\alpha, \beta_1, \ldots, \beta_n \notin R_u$ to ensure that u cannot destroy our secret

iv) if u can now execute a valid and authorised transaction with respect to D_u which, however, will be rejected by D, prevent it – try to find one or more aliases $\gamma_1, \ldots, \gamma_k$ such that, if we set

$$M_u(\Sigma_u)(\gamma_i) = \neg M(\Sigma)(\gamma_i)$$

for all $i = 1, \ldots, k$, then every valid and authorised transaction with respect to D_u is also accepted by D

v) to record all these distortions we must introduce an alias-log P:

a) insert into P

$P(D_u, \alpha, \text{secret}, \{\beta_1, \ldots, \beta_n, \gamma_1, \ldots, \gamma_k\})$

b) if some of the aliases

$\delta_1, \ldots, \delta_l \in \{\beta_1, \ldots, \beta_n, \gamma_1, \ldots, \gamma_k\}$

have required further aliases for themselves, then insert into P for all $i = 1, \ldots, l$

$P(D_u, \delta_i, \text{alias}, \{\delta^i_1, \ldots, \delta^i_{r_i}\})$

There are no changes to the signature Σ_u and to the integrity constraints C_u, ie $\Sigma_u = \Sigma$, $C_u = C$ and $D_u = (\Sigma, C, I_u)$ As long as we have a relational database, there is a simple correspondence between the content of the state I_u and the intended model $M_u(\Sigma)$.

Thus, we have constructed a function which represents the intended model of the CDB D_u,

$$M_u(\Sigma) : A_G(\Sigma) \to \{\text{True, False}\} \qquad (3)$$

which assigns to all tuples that do not occur in the alias-log P the same truth value as (2), the intended model of the CDB, does and which assigns to all tuples that occur in P, ie, which should be kept secret from u or are a consequence of this confidentiality demand, the opposite truth value than (2) does. In this sense we can say that (3) is a distortion of (2) and the parameter of the distortion is the assignment of truth values.

To draw a parallel to real life, we hide a secret by telling some more lies and pretend that everything is in perfect order – at the same time we are well aware of the possibility that other people behave in the same manner in front of us.

This type of confidentiality applies only to ground atomic formulae, ie tuples – we cannot change the truth value of a term, ie an attribute value, or a a relation because they, simply, do not have truth values. To include this type of confidentiality in our database we only have to include the alias-log

P into it. Thus, the open extended database with users and powers

$$D = (\Sigma, C, I, U, \{R_u\}_{u \in U}, \{K_u\}_{u \in U})$$

now becomes a database with confidentiality

$$D = (\Sigma, C, I, U, \{R_u\}_{u \in U}, \{K_u\}_{u \in U}, P) \quad (4)$$

To construct a particular distorted database D_u we take the model of D, apply the distortions recorded in the alias-log P to it and then translate the distortions of the model to distortions of the state of D_u.

EXAMPLE 3 Let us take a look at Example 1 and let us say that it represents the open database D. Suppose that u is a user and the tuple Missions(Enterprise, Asgard, Fun) should be kept secret from him. We now construct the distorted database $D_u = (\Sigma, C, I_u)$ with the distorted model $M_u(\Sigma)$. The Starships relation remains unaltered, ie, I_u comprises

Starships	Starship
	Enterprise
	Voyager

To reverse the truth value of the secret tuple in D_u we, firstly, present to u the following Missions relation:

Missions	Starship	Destination	Objective
	Voyager	Midgard	Aid

We can skip step (ii) since all integrity constraints remain valid. Suppose that the secret tuple is not in the update-rights of u. Now at step four, we notice that u has the powers to schedule the Enterprise to Muspelheim. If he decides to insert the tuple Missions(Enterprise, Muspelheim, Research), ie, the Missions relation in D_u would be as follows

Missions	Starship	Destination	Objective
	Enterprise	Muspelheim	Research
	Voyager	Midgard	Aid

then we have a violation of the dynamic semantics of D_u. The transaction of u is authorised and valid with respect to u and, consequently, must be accepted by D_u. Since D and D_u are not two separate databases but D_u is only a distortion of D, the transaction is immediately propagated to D and, obviously, it will be rejected by D – an action for which u cannot find any explanation in D_u. To avoid this situation we notice that u has no update rights on tuples that comprise Nifelheim. So, as an alias for the secret tuple Missions(Enterprise, Asgard, Fun), we insert into the Missions relation the tuple Missions(Enterprise, Nifelheim, Maintenance). The final Missions relation in D_u is then:

Missions	Starship	Destin.	Objective
	Enterprise	Nifelheim	Maintenance
	Voyager	Midgard	Aid

If u now tries to insert his tuple, this will be rejected and he can find an explanation for it in D_u: a violation of the primary key. To keep the example short we do not extend our consideration to the Freight relation.

Suppose that we have flattened the alias-log to a relation, then, lastly, we insert into it the tuple $P(D_u$,Missions(Enterprise,Asgard,Fun), secret, Missions(Enterprise, Nifelheim, Maintenance)). ◺

5.3.2 Distortion of the domain

An apparent shortcoming of the above-defined type of confidentiality is that we are unable to keep the existence of, eg, the starship Enterprise secret. Suppose that in the first step we remove the tuple Starships(Enterprise) from the Starships relation in the distorted database. But then, in step four, we notice that we need an alias to preserve the database semantics but there is none. Well, one can object that we can remove all tuples that comprise the name Enterprise from the user's update rights. But if he has the powers to buy new starships how do we justify in front of him the fact that he is not allowed to name a new starship Enterprise? This problem is quite important in practice since, eg, there may be a public hangar for the Voyager starship and a secret

hangar for the Enterprise. To distinguish this type of confidentiality from the previous one, we must note that the object of confidentiality of the previous type is the truth value of a statement. Here, the object of confidentiality is the existence of objects and properties.

A solution to this problem, structured name-spaces, is presented in Spalka/Cremers (1997). Here we show only how structured name-spaces can be interpreted as a distortion of the intended model.

We recall that real-world objects correspond to attribute values of a tuple or terms and properties to relations or predicates. In order to keep the existence of an object or a relation secret the database must pretend that it is not an element of the real-world section, ie, when asked about it a possible answer is 'I do not know what you are talking about'. If we again take a look at the intended model

$$M(\Sigma) : A_G(\Sigma) \to \{\text{True, False}\}$$

then we note that the previous type of confidentiality modifies the value of this function at certain points. Now, to pretend that some objects or properties are unknown, this function must become undefined at all points, which comprise the secret object or property. The obvious way to achieve it is to remove these points from the function's domain $A_G(\Sigma)$. Since $A_G(\Sigma)$ is determined by the signature Σ, we can modify it only by modifying Σ. Some of the previous works have shown that it is quite easy to present to a user u a database with a distorted signature $\Sigma_u = (\mathfrak{A}, P_u, \rho_u)$ from which some relations are removed, ie $P_u \subseteq P$. In this case the user is not told that, eg, there is a relation that stores a cargo's value, but he can still ask about Enterprise. To prevent him from doing this we must structure the database's flat name-space. To give an example, a file system can store two files with same name if they are in different directories. Here, the full path-name of the directory serves as a name-space selector and the name of the file is a local name within the selected name-space.

Spalka/Cremers (1997) come to the conclusion that following modifications to $\Sigma = (\mathfrak{A}, P, \rho)$, the signature, are necessary to obtain a database with structured name-spaces:

- $\mathfrak{A}^*$ is regarded as a universal name-space
- There is a new set E of name-space selectors
- $N = E \times \mathfrak{A}^*$ is the new global universal name-space and $(a, b) \in E \times \mathfrak{A}^*$ an element of it
- The sets of terms and predicates are now defined over N (and not over $\mathfrak{A}^*$)
- For each user $u \in U$ there is a set $E_u \subseteq E$ of name-space selectors available to u, which defines his personal universal name-space

$$N_u = \bigcup_{e \in E_u} \{e\} \times \mathfrak{A}^*.$$

At a first glance, we must extend our database with two components: E and the set of all E_u. However, Spalka/Cremers (1997) show that if confidentiality is the purpose for using structured name-spaces, then setting

$$E = \mathfrak{P}(U) \tag{5}$$

the power-set of U, already offers maximum flexibility.* We therefore assume (5) and give the final definition of a database with confidentiality:

$$D = (\Sigma, C, I, U, \{R_u\}_{u \in U}, \{K_u\}_{u \in U}, P, \{E_u\}_{u \in U}) \tag{6}$$

We just need one more word on P. The intended model of D is now the function

$$M(\Sigma) : A_G(\Sigma) \to \{\text{True, False}\} \tag{7}$$

such that $A_G(\Sigma)$ is defined over N, and the intended model of a distorted database D_u is now the function

$$M(\Sigma_u) : A_G(\Sigma_u) \to \{\text{True, False}\} \tag{8}$$

such that $A_G(\Sigma_u)$ is defined over N_u. If we begin the construction of (8) by reducing (7) to $A_G(\Sigma_u)$, then we need to store in the alias-log P only the reversed truth values with respect to $A_G(\Sigma_u)$.

EXAMPLE 4 Let us again take a look at Example 1. Suppose that the starship Enterprise is hidden in a secret hangar and should be kept secret from the user u. Firstly, we introduce a structure on the database's flat name-space. We set

$$E = \mathfrak{P}(U) = \{\varnothing, \{u\}\}$$

and $E_u = \{\{u\}\}$. Then the Starships relation re-

* It is also sufficient to model multi-level structures.

ceives the following representation in D:

({u}, Starships)	Starship
	($\varnothing$, Enterprise)
	({u}, Voyager)

Since $N_u = \{u\} \times \mathfrak{A}^*$, the reduction of (7) to (8) with respect to N_u yields the following Starships relation in D_u:

({u}, Starships)	Starship
	({u}, Voyager)

Now, if u buys a new starship and decides to name it Enterprise he can enter without any problems the tuple ({u}, Starships) ({u}, Enterprise) into the database.

Suppose we wish to keep the Freight relation secret from u, then we only need to assign to it a name-space selector which is unavailable to u, ie, we set ($\varnothing$, Freight). ◺

5.3.3 Some remarks

In (6) we have arrived at a database that supports two forms of confidentiality. It is a well defined database in the following sense:

i) Its static semantics is identical to that of an open database, viz, both the open database and any distortion thereof have a unique intended model

ii) Its dynamic semantics is a natural restriction of that of an open database since a transaction must be both valid and authorised to be accepted*

iii) It defines the meaning of confidentiality in a precise declarative manner and there is a clear correspondence to the real life

iv) As in real life it rejects confidentiality demands that are recognised to be not satisfiable

A consequence of point (i) is that we do not even need to think of polyinstantiation. Since we have a clear distinction between the ODB and any CDB, we always precisely know the intended state of affairs in the OWS and we have full control of the distortions in any CWS.

A consequence of the points (i), (ii) and (iv) is that we also do not need to think of inference. With respect to our approach, inference can be defined as the ability of a user of a CDB to discover one or more distortions and, thus, infer its intended state in the ODB. However, the way we have constructed (6) precludes exactly this possibility. Well, of course, the user of the CDB may know somebody who knows something about a distorted element – that's life, but what is more important is that our database cannot do anything about it. It can take into consideration only those elements of which it knows – and these are precisely stated in (6).

We have identified five forms of confidentiality and shown that one of it cannot be influenced by the database, two of them are not compatible with the CWA and the final two forms can be integrated in a database. There is probably no formal proof that there are no other forms of confidentiality but if we identify the semantics of a database with its intended model, then there are no other ways of distorting a function. The only additional distortion we can apply is to include alias-predicates in the distorted domain, ie, the CDB comprises relations that do not exist. However, we do not regard this distortion as a form of confidentiality but only as a tool for the satisfaction of confidentiality demands.†

Lastly, this paper gives only a definition of a database. The important question of how we determine whether a confidentiality demand is satisfiable and, if it is, how we compute the necessary distortions, is left open. We will present an answer to it in the near future.

6 Conclusion

The motivation for this work is the question of how

* However, it is much harder to define and compute the effect of a transaction if the users are embedded in a hierarchy of groups. We will soon report on this problem in a paper.

† We presently investigate whether there are situations in which the inclusion of false predicates is necessary.

we can introduce real-life confidentiality in an open database in a declarative manner. In a first step we have introduced users and their powers in the open database and argued that the powers users have in real life must also be respected by the database, which implies that there is a set of clearly not satisfiable confidentiality demands. We have then presented five forms of real-life confidentiality, we have translated them in the context of logic and shown that only two forms can be brought in accord with the Closed World Assumption in our database. These two forms, which correspond to real-life situations in which we do not want to admit that we are trying to keep something secret, can be precisely defined as single-model distortions of the database's intended model. Our view that confidentiality must be seen as a distortion of an open world-section has led us to a database with confidentiality, which is an extension to open databases, ie, it possesses unequivocal static and dynamic semantics. A consequence of this and some other properties is that we never encounter polyinstantiation and that there is no inference. The most important result of this work is the final definition of a database with confidentiality. We have shown that the components of this database are necessary in order to introduce real-life confidentiality to an open database.

In the near future we will present more details on the definition and execution of transactions in our database and some methods for the enforcement of confidentiality demands.

Lastly, we would like to make a remark on a problem that is closely related to confidentiality, though not an issue of the database design. In a database that supports confidentiality, users and programs can receive incomplete or wrong information. This distorted information is often used in further decisions. We believe that at this stage it is the responsibility of users who state confidentiality demands to contemplate on their negative consequences and implications.

Acknowledgements

We would like to express our sincere thanks to the anonymous referees who have helped to amend this work.

References

Bonatti, Piero, Sarit Kraus and V.S. Subrahmanian. (1992) 'Declarative Foundations of Secure Deductive Databases'. Ed Joachim Biskup and Richard Hull. *4th International Conference on Database Theory – ICDT'92.* LNCS, vol 646. Berlin, Heidelberg: Springer-Verlag. pp 391–406. [Also in: *IEEE Transactions on Knowledge and Data Engineering* 7.3 (1995):406–422.]

Cremers, Armin B., Ulrike Griefahn and Ralf Hinze. (1994) *Deduktive Datenbanken.* Braunschweig: Vieweg.

Das, Subrata Kumar. (1992) *Deductive Databases and Logic Programming.* Wokingham, England: Addison-Wesley.

Denning, Dorothy E., Teresa F. Lunt, Roger R. Schell, Mark Heckman and William R. Shockley. (1987) 'A Multilevel Relational Data Model'. *1987 IEEE Symposium on Security and Privacy.* IEEE Computer Society Press. pp 220-234.

Landwehr, Carl E. (1981) 'Formal Models for Computer Security'. *ACM Computing Surveys* 13.3:247-278.

Morgenstern, Matthew. (1987) 'Security and Inference in Multilevel Database and Knowledge-Base Systems'. *1987 ACM SIGMOD Conference / SIGMOD Record* 16.3:357-373.

Morgenstern, Matthew. (1988) 'Controlling Logical Inference in Multilevel Database Systems'. *1988 IEEE Symposium on Security and Privacy.* IEEE Computer Society Press. pp 245-255.

Reiter, Raymond. (1984) 'Towards a Logical Reconstruction of Relational Database Theory'. Ed Michael L. Brodie, John Mylopoulos and Joachim W. Schmidt. *On Conceptual Modeling.* New York: Springer-Verlag. pp 191-238.

Sicherman, George L., Wiebren de Jonge and Reind P. van de Riet. (1983) 'Answering Queries Without Revealing Secrets'. *ACM Transactions on Database Systems* 8.1:41–59.

Spalka, Adrian, and Armin B. Cremers. (1997) 'Structured name-spaces in secure databases'. Ed T. Y. Lin and Shelly Qian. *Database Security XI.* IFIP TC11 WG11.3 Conference on Database Security. London at al: Chapman & Hall, 1998. pp 291-306.

Thuraisingham, Bhavani M. (1991) 'A Nonmonotonic Typed Multilevel Logic for Multilevel Secure Data/Knowledge Base Management Systems'. *The Computer Security Foundations Workshop IV.* IEEE Computer Society Press. pp 127-138.

—. (1992) 'A Nonmonotonic Typed Multilevel Logic for Multilevel Secure Data/Knowledge Base Management Systems – II'. *The Computer Security Foundations Workshop V.* IEEE Computer Society Press. pp 135-146.

Temporal Authorization Models Session

Wednesday, July 28, 1999

Chair: David Spooner
Rensselaer Polytechnic Institute

Temporal Authorisation in the Simplified Event Calculus

Steve Barker
Cavendish School of Computer Science
University of Westminster
London, U.K, W1M 8JS

Abstract

This paper shows how the simplified event calculus (SEC) may be used as the basis for representing security models for discretionary access control when access rights may be expressed as holding for limited periods of time.

The approach presented here involves using normal clause logic to write a set of axioms to represent a specific security model with time constrained authorisations. These model specific axioms are combined with a set of rules which represent the core axiom of the SEC and a set of ground atomic assertions which represent a history of security events which affect a database. In this framework, a subject's request to access a database object is allowed only if the access request can be proved to hold from the resulting axiomatisation.

An example security model is presented to demonstrate the approach, extensions to this model are outlined and some important practical issues are discussed.

1 Introduction

The need to protect a database against unauthorised access has long been recognised as important [7]. However, whilst languages for expressing access rights and methods for deciding questions of authorisation have been supported in commercial database management systems for some time, more powerful means for expressing security information are increasingly being demanded. One such demand is for access control methods which permit privileges on database objects to be granted to subjects for limited periods of time [18].

When access rights are only allowed for a certain duration they are automatically removed on the expiration of the interval of time for which the rights were initially permitted. The facility which enables access to be specified as lasting for a restricted interval of time gives the grantor of a right a good deal of control over the permissions they give to grantees and restricts the scope the latter have for compromising the security of the data contained in a database.

There are many practical situations in which time-constrained access rights are required. In our own computing department, for instance, it has proved necessary to grant privileges on resources to students for different periods of time depending on such things as their enrolment status and the type of course they are attending. Thus far, this facility has been supported via ad-hoc, low-level routines ; a more general, higher-level approach is, however, needed.

This paper presents a method which may be used for implementing a system for discretionary access control when access rights may be restricted by time. As such, our work addresses similar issues to those considered in Bertino *et al.* [3]. The approach we adopt is, however, quite different and has certain distinct advantages.

In [3], a language is described which enables users to write *temporal authorisations* and *derivation rules.* The former are used to explicitly record the rights individuals have been granted on objects by whom and for how long. The latter are used to derive the privileges which are implied by the temporal authorisations. Both the temporal authorisations and the derivation rules may be expressed as holding for a particular interval of time.

Rather than providing a specific language for users to express their security information, in our treatment a security administrator chooses a security model to implement and formulates this model in the *simplified event calculus (SEC)* [12]. Moreover, the notion of an event, rather than an interval of time, is central in our approach.

We argue that the expressive power of the SEC provides a security officer with considerable scope for representing various security models which support the specification of time-constrained privileges. This scope is achieved by treating security actions as constants in rules which are written in *clausal form logic* to describe the consequences of a security event be-

ing performed on a database. The use of clausal form logic, on which the SEC and our approach is based, makes it possible for users to express their security requirements in a clear and concise way. Moreover, this formalisation may be regarded as an executable specification which may be verified, informally with prospective end-users and formally by proving properties of the specification, prior to implementation.

Our approach has the additional attraction of enabling users to easily express authorisations by using a set of ground atomic binary relations. A simple front-end may be straightforwardly implemented to further ease the burden of representing security information.

The rest of this paper is organised in the following way. In Section 2, a brief overview of the simplified event calculus is presented. In Section 3, the details of the event-based specification of security information are given. In Section 4, our use of the simplified event calculus to represent a particular security model with temporal authorisations is outlined. In Section 5, theoretical and practical issues are addressed and, in Section 6, various extensions to our example security model are discussed. In Section 7, some conclusions are drawn and suggestions are made for further work.

2 The Simplified Event Calculus (SEC)

The original *event calculus* [8] consists of a number of general axioms, expressed in terms of Horn clauses augmented with *negation as failure* [4], which are intended to be used for knowledge representation in situations where reasoning about time and events is required.

The basic idea is that the general axioms of the event calculus be combined with a set of domain specific axioms, which define the initiation and termination of relationships, and a description of events which have occurred in the world a database purports to describe. From this set of axioms and a description of events the consequences of those events may be derived together with the periods of time for which these consequences hold.

The simplified event calculus, as its name suggests, is a restricted form of the event calculus which nonetheless has proved to be useful for treating a number of practical problems ([5], [8], [16]).

The SEC only permits forward persistence and is based on the simplifying assumption that complete information exists about events, including the times at which events occur. Under this assumption, a single persistence axiom is all that is required to specify that an initiated relationship, R, continues to persist until an event occurs to terminate it. The core axiom of the SEC, *C0* (say), which captures this notion may be expressed thus (where $\neg$ denotes a chosen form of negation and $<$ is an "earlier than" relation) :

$$
\begin{aligned}
holdsat(R,T) \quad \leftarrow \\
& happens(E1,T1), \\
& initiates(E1,R), \\
& T1 < T, \\
& \neg\exists E2,T2 \\
& [happens(E2,T2), \\
& terminates(E2,R), T1 < T2]
\end{aligned}
$$

More fully, *C0* expresses that a relationship R holds at a time point T if an event *E1* happened which initiated R (ie. made R true) at an earlier point in time *T1* and no intervening event *E2* has terminated R (ie. made R false) subsequent to its initiation.

Notice that for *C0*, and henceforth, terms which appear in the upper case denote variables.

An example of the type of domain specific rules which are used to define the *initiates* and *terminates* predicates appearing in *C0* is given in Section 4. Before presenting these rules, however, we consider first the representation of a history of security events performed on a database.

3 Event Descriptions and Authorisation Histories

To record the fact that a security event has taken place, the notion of a *security event description* is used. A security event description consists of a set of ground assertions which, as the name implies, describes the occurrence of an event which has taken place and which is relevant to the security of a database. To see what is involved in representing these events, suppose that Bob creates a database object o1 on 1/1/99 and takes read and write access rights on o1. This event, *e0* (say), may be represented by the following event description :

happens(e0,1/1/99),
act(e0,create),
creator(e0,bob),
object(e0,o1),
mode(e0,read),
mode(e0,write).

In keeping with the language used in the event calculus, the choice of predicate names in the assertions appearing in the event description for *e0* are influenced by the work on *semantic cases* ([6], [13]). Any

constant which appears as an argument in one of these predicates is one of : an event, a time, a database subject, a group identifier, a database object or an action which is permissible in the security model.

In the event *e0*, *create* is the action which is performed. Now, whilst a subject clearly needs to be able to do more that just create objects, the precise set of actions a subject may perform will depend on the choice of security model to be implemented.

The actions upon which the example security model given in Section 4 is based are those which are included in the set, A={create, grant, grantgroup, revokeright, revokegroupright, destroy} ; the access modes considered are those in the set, M={read, write}. That is, in this security model, subjects may create and destroy objects and may grant and revoke some or all of the read and write privileges individuals or groups of individuals may have on database objects.

Notice that the security event descriptions for all of the actions in A will be similar in form to the event description for *e0*. Whilst the predicate names and constants will differ, a security event description will always be represented using a set of ground binary relations. A simple front-end may be used to both assist users in describing security events and to ensure that security event descriptions are syntactically and semantically meaningful.

To see more fully what is involved in describing a history of events affecting database security, suppose that Bob has created the database object, o1, and consider the following narrative :

> On 2/1/99 Bob grants John write access on o1 until 5/1/99 and read access until 20/6/99. On 15/4/99, Bob grants Sue read and write privileges on o1 indefinitely into the future. On 25/4/99, Bob grants every member of the Sales department read access to o1 until 1/6/99. Sue's write privilege on o1 is removed, by Bob, on 20/5/99.

The corresponding set of event descriptions describing the narrative above (in which ei, where i is a natural number, denotes an event) is as follows :

happens(e1,2/1/99),
act(e1,grant),
grantee(e1,john),
object(e1,o1),
mode(e1,write),
stop(e1,5/1/99).

happens(e2,2/1/99),
act(e2,grant),
grantee(e2,john),
object(e2,o1),
mode(e2,read),
stop(e2,20/6/99).

happens(e3,15/4/99),
act(e3,grant),
grantee(e3,sue),
object(e3,o1),
mode(e3,read),
mode(e3,write).

happens(e4,25/4/99),
act(e4,grantgroup),
grantee(e4,sales),
object(e4,o1),
mode(e4,read),
stop(e4,1/6/99).

happens(e5,20/5/99),
act(e5,revokeright),
revokee(e5,sue),
object(e5,o1),
mode(e5,write).

A set of security event descriptions is recorded as part of the information which is maintained in the database to be used for access control. In the example security model which we consider, only the creator of a database object, O, can grant and remove access rights and decide the period of time for which rights are allowed on the object. As such, an event description is only permissible for O if it is asserted by the creator of O. A validation procedure for security event descriptions ensures that the specifier of such a description for O is always O's creator.

Henceforth, we will refer to a set of security event descriptions, like the one above, as an *authorisation history.*

Notice that, in our example security model, it is the grant and grantgroup operations which have a temporal component to them. That is, rights may be specified as being granted for a limited period of time. Not surprisingly, the removal of a privilege, P, from a subject, S, on an object, O, at a time, T, prevents S gaining P access to O unless and until S is granted the privilege P at some time point which is subsequent to T.

An access right is assumed to hold from the point in time at which the event which grants the access

actually happens. The difference between the time in a *happens(e,t)* fact and a *stop(e,t1)* fact (where t < t1) is the interval of time for which the right, granted in the event e, is permitted to hold for the subject(s) and object specified in a security event description. As our example authorisation history demonstrates, if a stop time is not associated with a grant then the right which is granted is assumed to be permitted indefinitely into the future. Conversely, if a stop time is associated with a grant then that specifies the maximum future time point up until which a privilege is to hold. The grantor of a privilege has the option of terminating this privilege earlier than the maximal time by using a revoke operation from A.

4 Formulating a security model in the SEC

As we have said, there are two sets of rules which are important in our approach. One set of rules is needed to represent the axiom, *C0*, of the SEC ; the other set is needed to define the initiation and termination of the properties of interest in the specific security model, with time constrained access rights, to be implemented.

To simulate the *C0* axiom the following three rules may be used :

```
(C1)
 holds(access(S,P,O))   ←
                          happens(E,T),
                          initiates(E,access(S,P,O)),
                          not ended(E,access(S,P,O),T).

(C2)
 ended(E,access(S,P,O),T)   ←
                              happens(E1,T1),
                              T < T1,
                              terminates
                                (E1,access(S,P,O)).

(C3)
 ended(E,access(S,P,O),T)   ←
                              stop(E,T1),
                              current-time(T2),
                              T1 < T2.
```

The reading of this set of rules is similar to that for *C0* : if an event, E, happens at time, T, which causes a subject, S, to be given the access right, P, on a database object, O, and the period of time for which this right was granted has not expired and no subsequent event is known to have occurred to have ended the right P which S has to access O at time T then S holds the privilege P on O. The variable, T2, in the *current-time* atom is instantiated with the time, taken from the "system clock", at which the *current-time* atom in *C3* is selected in the process of deciding whether a privilege has expired.

Notice that the *C2* rule deals with the termination of a privilege as a consequence of the occurrence of an event whereas *C3* deals with the termination of a privilege as a consequence of the ending of a period of time for which the privilege was granted. Note also that the T < T1 condition in the *C2* rule assumes that properties hold only after their initiating event happens. The T1 < T2 condition in *C3* is based on the assumption that event descriptions do not have identical times for the *happens* and *stop* predicates (any attempt to include this type of meaningless security event description in an authorisation history would be rejected in the process of validating event descriptions).

To see what is involved in writing the domain-specific *initiates* and *terminates* rules for the example security model based on A, we consider first the *initiates* rule, *P1*, required to treat object creation. For that, suppose that an event, E, happens in which a subject, S, creates an object, O, and takes the right, P, on O. This event has the consequence of initiating (ie. making true) the fact that S has the right P on O viz. :

```
(P1)
 initiates(E,access(S,P,O))   ←
                                act(E,create),
                                creator(E,S),
                                object(E,O),
                                mode(E,P).
```

Similarly, if the creator of object O grants subject S the right P on O then the initiation of a period of time during which S may exercise the privilege P on O may be represented by the following rule :

```
(P2)
 initiates(E,access(S,P,O))   ←
                                act(E,grant),
                                grantee(E,S),
                                object(E,O),
                                mode(E,P).
```

To deal with the granting of rights to groups of individuals the body of the rule *P2* needs to be slightly modified to include a condition for testing for group membership thus :

(P3)

initiates(E,access(S,P,O)) ←
act(E,grantgroup),
grantee(E,W),
memberof(S,W),
object(E,O),
mode(E,P).

Notice that a set of *memberof* facts (eg. *memberof(bill,sales)*) may be recorded in the database to permit group authorisations to be specified (the special group, *public*, may also be used to grant privileges globally) and subjects may be members of several different groups simultaneously (without, as we will see later, conflicting privileges arising).

When combined with an authorisation history and the core axioms of the SEC, the *initiates* rules are used to infer the start of an interval of time for which access rights are held by subjects on database objects.

As previously intimated, for the situations in which the interval of time for which an access right holds is ended by a *stop* time in an event description the *C3* axiom is used. For the cases where an action is performed which ends the right a subject has to access an object, a set of *terminates* rules is required. These *terminates* rules are used in conjunction with the core axiom *C2* and enable an owner of an object to remove privileges from subjects.

The following pair of *terminates* rules may be used for dealing with revocations in our example model :

(P4)

terminates(E,access(S,P,O)) ←
act(E,revokeright),
revokee(E,S),
object(E,O),
mode(E,P).

(P5)

terminates(E,access(S,P,O)) ←
act(E,revokegroupright),
revokee(E,W),
memberof(S,W),
object(E,O),
mode(E,P).

Finally, to deal with the termination of rights when an object is destroyed by its creator we may write :

(P6)

terminates(E,access(,*,O))* ←
act(E,destroy),
object(E,O).

In *P6* the variable '*' denotes any subject and any privilege.

As we have said, the set of access rules, *Pi* ($i \in \{1..6\}$), represents the particular security model which we have chosen to use to demonstrate our approach. Different security models may be represented by choosing different security actions to support and then writing the model-specific axioms to represent the consequences of performing these actions. Similarly, any number of auxiliary rules may be included in a security model to simplify the specification of an authorisation history (for example, a "write permission implies read permission" rule may be used).

Notice that, unlike the specifications in [3], the rules defining the security theory above do not have a time interval associated with them to represent the periods during which they are applicable to a particular subject exercising a right to access an object. In our approach, the intervals of time for which the rules are applicable to a (subject,privilege,object) triple are implicit from the times included in an authorisation history. This simplifies the specification of time-constrained privileges but without compromising the scope a security officer has for representing security theories.

5 The Access Control Procedure

Given an authorisation history, to decide whether a subject, S, should be permitted to exercise the privilege, P, on object, O, at the time at which access is requested, the approach we adopt involves attempting to prove that the access request is a theorem of the authorisation history together with the axiomatisation which defines a chosen security model with time-constrained privileges. The existence of a proof permits S to perform P on O at the time at which the access is requested ; otherwise the access request is denied.

Henceforth, we will denote, by Σ, the set of core axioms, *Ci* ($i \in \{1..3\}$), together with the *Pj* rules, ($j \in \{1..6\}$), which define our example security model. Similarly, we will denote the authorisation history given in Section 3 by **H**.

Since Σ can be described as a set of function-free *stratified clauses* [1] it follows that questions of authorisation may be answered of this security theory by using *safe SLDNF-resolution* [10]. That is, to decide whether a subject, S, can exercise privilege, P, on a object, O, an *SLDNF-derivation* for *holds(S,P,O))* may be attempted on the security theory $\Sigma \cup \mathbf{H}$ to decide whether to permit the access request or not. In this context, if $\Sigma \cup \mathbf{H}$

$\cup$ {$\leftarrow$holds(access(S,P,O))} has an *SLDNF-refutation* then subject S has the privilege P on O. Conversely, if $\Sigma \cup \mathbf{H} \cup$ {$\leftarrow$holds(access(S,P,O))} has a *finitely-failed SLDNF-tree* then S does not have the privilege P on O.

Notice that it is impossible for inconsistent access rights to arise in Σ for any authorisation history. For inconsistency, *holds(access(S,P,O))* would need to be simultaneously true and false (for some substitution for variables). Since, however, there is only one definition of *holds* in Σ and SLDNF-resolution is consistent, it should be clear that inconsistent access rights do not arise.

Before giving an example of the use of SLDNF-resolution to decide questions of access, we need to mention an important practical issue. For efficiency reasons, ground instances of the *initiates*, *terminates* and *ended* predicates may be *memoised* [11], by dynamic assertion [17], when a *holds(access(S,P,O))* request is first evaluated. This approach, corresponding to the use of "view materialisation" in [3], increases the size of the security theory but has the advantage of permitting proofs of authorisation to be efficiently performed since ground assertions may be used, almost entirely, in the process of deciding questions of access. The existence of previously generated *initiates*, *terminates* and *ended* facts is assumed in the example, of the use of the access control method, which follows.

Example
From the theory, Σ, and the authorisation history, **H**, the following SLDNF-tree (which assumes a *leftmost selection rule* is in force [10] and, for brevity, omits the computation for *not ended* and the alternative computations which follow from $\leftarrow$*happens(E,T)*) shows that on 25/1/99 (say) John does not have a write privilege over o1 :

```
<-holds(access(john,write,o1))
                |
<-happens(E,T),
  initiates(E,access(john,write,o1)),
   not ended(E,access(john,write,o1),T)
                |
<-initiates(e1,access(john,write,o1)),
    not ended(e1,access(john,write,o1),2/1/99)
                |
<-not ended(e1,access(john,write,o1),2/1/99)
              fail
```

The SLDNF-tree for $\Sigma \cup \mathbf{H} \cup$ {$\leftarrow$holds(access(John,read,o1))} is very similar to the one above. However, whereas *ended* succeeds by *C3* for the *holds(access(john,write,o1))* case, in the test for John's possession of a read right on o1 the *ended* subgoal finitely fails since John's read privilege over o1 has been neither terminated by an event nor stopped as a consequence of the inclusion of a *stop* atom in an event description. Thus, an SLDNF-refutation exists for $\Sigma \cup \mathbf{H} \cup$ {$\leftarrow$holds(access(John,read,o1))}.

It immediately follows from the above that if John were to request write access to o1 on 25/1/99 then that request will be denied but a read request would be allowed.

An issue which is sometimes raised in the context of temporal authorisation models relates to the ending of a period of time for which a privilege has been granted *during* the test to determine whether a subject's access to a database object is permitted or not. In our view this is a matter of practical concern rather than a security *model* issue and is not specific to temporal authorisation models since, for instance, the revocation of a privilege can take place whilst a question of access is being decided in a non-temporal environment. The problem does not, however, arise in our approach. To see why notice that, since we assume that a leftmost selection rule is used in the SLDNF-derivations performed on Σ, in the *C3* rule, the current time is used immediately prior to the test for the expiration of a time-constrained access right. Hence, even if a time-constrained privilege does expire after the access control procedure has been invoked, it is not until the T1 < T2 condition is evaluated in *C3* that the expiration of a time-constrained privilege is checked and at this point in the computation the current time will have been extracted from the system clock immediately prior to the test for T1 < T2.

In implementations of our approach periodic garbage removal may be performed on the memoised assertions and on an authorisation history (to remove redundant security event descriptions). For instance, if the current time is subsequent to the *stop* time included in a security event then the event description may be physically deleted from the authorisation history together with any facts which were initiated by the event. For instance, the event description for e1 could be removed from **H** after 5/1/99 since John does not have write access on o1 after this point in time. If this deletion were performed then, in any subsequent test of John's permission to write o1, the access control procedure may establish more quickly that John does not have this privilege since the early

failure of the *initiates(E,access(john,write,o1))* condition will cause *holds(access(john,write,o1))* to fail early too. Similarly, when an object is destroyed the event description which created that object may be deleted together with any dynamically asserted *initiates*, *ended* or *terminates* facts relating to the object. Other optimisations which are possible include changing the order of the conditions appearing in the rules of Σ to exploit the subgoal selection strategy used in an SLDNF-derivation. For example, by resolving an *initiates(E,access(S,P,O))* subgoal with an appropriate memoised ground instance of the same form, it is possible to delay selecting the non-ground *happens(E,T)* condition in *C1* until substitutions for E and T have been found.

Another practical issue which needs to be considered relates to the *soundness* and *correctness* results applicable to the proof method used to decide questions of access. As we have seen, for our security model the authorisation test uses SLDNF-resolution. As such, for Σ (together with any authorisation history), Clark's *2-valued completion* [4] is a candidate declarative semantics for our approach. In fact, since Σ is an *allowed* [14] and *hierarchical* [4] normal clause theory and *holds(access(S,P,O))* is an *allowed query*, it follows, from [15], that, for Σ, (safe) SLDNF-resolution is sound and complete with respect to Clark's semantics for determining the privileges subjects have on database objects.

In this context, the soundness of SLDNF-resolution implies that SLDNF-resolution correctly identifies which subjects have which access rights on which database objects. Conversely, completeness means that all the access rights which are implied by an authorisation history together with Σ can be derived using SLDNF-resolution.

The practical issue which arises from the discussion on soundness and completeness is that an implementor needs to consider the technical results which an available theorem prover possesses when formulating a security model using the approach we advocate.

6 Extending Σ

It should be clear from the discussion above that a significant attraction of our approach is that it permits any number of security theories with temporal authorisations to be expressed in a simple and completely uniform way. For a different security theory to Σ, all that is required is that a different set of actions to those contained in A be chosen and that the *initiates* and *terminates* rules which define the consequences of performing these actions be formulated.

In this section, we consider some possible extensions to Σ to demonstrate the flexibility afforded by our approach. More specifically, we consider how *negative authorisations* may be expressed, how shared privileges may be specified, how *proactive authorisations* can be supported and how rules for default authorisations, like those in [3], may be represented.

To permit negative authorisations we view a denial as terminating the possibility of a subject accessing an object O to perform operation P on O. In this case, the required *terminates* rule (*C7* say) is as follows :

terminates(E,access(S,P,O)) $\leftarrow$
act(E,denying),
subject(E,S),
object(E,O),
mode(E,P).

This rule enables the owner of an object O to deny S the privilege P on O by including a suitable security event description in an authorisation history. For example, to represent that (as of 1/1/99) Sam is to be denied write access to o1, the following event description could be included in an authorisation history by the creator of o1 :

happens(e6,1/1/99),
act(e6,denying),
subject(e6,sam),
object(e6,o1),
mode(e6,write).

By inspection of our representation of the core axioms of the SEC, it will be clear that if S is denied the privilege, P, on O via an event description, Δ, in an authorisation history, Π, and *holds(access(S,P,O))* is not a theorem of a security theory, Γ which includes the *C7* rule and the core axioms of Σ then *holds(access(S,P,O))* is not provable by SLDNF-resolution from $\Gamma \cup \Pi$ whilst Δ is part of Π. This follows since, by *C7*, *terminates(E,access(S,P,O))* is a theorem of $\Gamma \cup \Pi$ where E is substituted with the event identifier for Δ. Similarly, if *holds(access(S,P,O))* is a theorem of Γ at time T1 and S is subsequently denied the privilege, P, on O at time T2, as a result of Δ being included in an authorisation history Ψ (say), then, from *C1*, *holds(access(S,P,O))* is not provable (using SLDNF-resolution) from $\Gamma \cup \Psi$, after T2, since *ended(E,access(S,P,O),T)* is a theorem of $\Gamma \cup \Psi$ using *C2* together with *C7*.

It follows that a *denials take precedence over permissions* meta-rule is effectively enforced by our par-

ticular formulation of the core axioms of the SEC. More specifically, this representation enforces a *terminates takes precedence over initiates* conflict resolution strategy.

By specifying authorisation histories which include denials of access, it is possible to express excepted authorisations in our approach. For instance, it will be clear from the above that to specify authorisations like "all members of marketing can read object o2 except Sam", what is required is that a pair of security event descriptions be included in an authorisation history ; one which grants read access to marketing whilst the other records that Sam's access to o2 is denied.

To permit negative authorisations to be specified as holding for a restricted interval of time then a *not denied(access(S,P,O))* condition may be added to the *C1* axiom whilst the following rule is added to the core axioms :

denied(access(S,P,O)) ←
happens(E,T1),
act(E,denying),
subject(E,S),
object(E,O),
mode(E,P),
stop(E,T2),
currenttime(T3),
T3 < T2, T1 < T3.

That is, in addition to access being prohibited as a result of the occurrence of an event which terminates a privilege or the expiration of a time interval for which a right was granted, a subject S may also be prevented from exercising the privilege P on O if S has been denied access to O at time T1 and the period of time for which the denial applies has not expired.

Proactive initiation of authorisations is also possible in our approach. To achieve that, an event description may be included in an authorisation history with an instance of a *happens(E,T1)* assertion such that if T is the current time then T < T1. As such, it is possible to specify on 1/6/99 that Sal will be authorised to read o1 from 1/10/99 (say).

To specify that privileges may be shared for a specified interval of time, the following rule may be used :

initiates(E,access(S,P,O)) ←
act(E,sharing),
subject(E,S1),
sharer(E,S),
object(E,O),
mode(E,P),
holds(access(S1,P,O)).

This rule expresses that the subject S may exercise the privilege P on object O for the period of time during which subject S1 has the privilege P on O. The *sharing* rule may be used with an event description like the following which records that, as of 3/3/99, Steve has read access on o1 if and as long as John has this privilege :

happens(e7,3/3/99),
act(e7,sharing),
subject(e7,john),
sharer(e7,steve),
object(e7,o1),
mode(e7,read).

In the authorisation model presented in [3], some authorisations may be derived as a consequence of the absence of another authorisation. That is, authorisations "by default" are possible. For example, in the language in [3] it is possible to use a *whenevernot* operator to express that subject S1 is permitted to exercise privilege P on O at all time points at which S2 does not hold the privilege P on O. This type of specification is straightforwardly expressible in our approach using the following axiom (*C8* say):

initiates(E,access(S1,P,O)) ←
act(E,defaultpermit),
permitted(E,S1),
excluded(E,S2),
object(E,O),
mode(E,P),
not holds
(access(S2,P,O)).

Conversely, the next rule enables the specifier of a security model to express that at all points in time at which subject S1 holds a privilege P on object O, subject S2 is prevented from exercising the P privilege over O :

terminates(E,access(S2,P,O)) ←
act(E,exclusion),
permitted(E,S1),
excluded(E,S2),
object(E,O),
mode(E,P),
holds(access(S1,P,O)).

The key point which arises from this discussion is that our approach provides a security administrator with a great deal of flexibility when it comes to spec-

ifying a security theory with time-constrained access rights. In principle, an implementor can choose to support any set of security actions and any set of privileges ; all that is required is that appropriate *initiates* and *terminates* rules be written to define the consequences of performing the selected security actions on a database. In contrast, pre-specifying a set of security operators to be supported precludes the formulation of security models which do not include these operators and, hence, limits the scope a security administrator has for protecting the data in a database.

The only constraints on a security administrator who uses our approach are those which apply to the methods used to decide whether a subject S has privilege P over O ; whether, for instance, the methods available for deciding questions of access are sound and complete (and with respect to what semantics).

7 Summary and Conclusions

By using the simplified event calculus it is possible to represent a range of security models for discretionary access control where privileges over database objects may be granted to subjects for limited periods of time. These security models may be verified to ensure that they satisfy organisational and technical acceptability criteria and may be formulated in subsets of clausal form logic for which efficient, sound and complete theorem provers are known to exist.

The "high-level" nature of the language on which the event calculus is based and the fact that users provide security information by writing ground atomic assertions makes the suggested approach especially easy to use. Moreover, a simple front-end may be developed to further ease the burden of expressing security information.

Security models which are directly based on the theoretical framework outlined above have thus far been implemented (in PROLOG) and have been successfully employed to manage access to relational databases. However, since no assumptions have been made about the underlying data model, the approach may also be used with other types of database.

In future work we will show how the SEC may be used to formulate a RBAC security model with time-constrained privileges and how our current work on security in deductive databases [2] can be extended to include temporal authorisations.

Acknowledgments

The author thanks Marek Sergot, Nigel Martin and Colin Myers for their encouragement and advice.

References

[1] Apt, K., Blair, H., and Walker, A., Towards a theory of declarative knowledge, in J. Minker(Ed.), *Foundations of Deductive Databases and Logic Programming*, Morgan-Kaufmann, CA, 1988.

[2] Barker, S., Security and integrity management for deductive databases. *In preparation.*

[3] Bertino, E., Bettini, C., Ferrari, E., and Samarati, P., A temporal access control mechanism for database systems, *IEEE Trans. on KDE*, 8(1), 1996.

[4] Clark, K., Negation as failure, in H.Gallaire and J. Minker(Eds), *Logic and Databases*, Plenum, NY, 1978.

[5] Eshghi, K., Abductive planning with the event calculus, *Proc. of ICLP*, MIT Press, 1988.

[6] Fillmore, C., The Case for Case, in Barch and Harms (Eds), *Universals in Linguistic Theory*, Holt, Reinhart and Winston, Chicago, 1968.

[7] Griffiths, P., and Wade, B., An authorisation mechanism for relational database systems, *ACM TODS*, 1, 3, 1976.

[8] Kowalski, R., Database updates in the event calculus, *Journal of Logic Programming*, 12, 1992.

[9] Kowalski, R. and Sergot, M., A logic-based calculus of events, *New Generation Computing*, 4(1), 1986.

[10] LLoyd, J., *Foundations of Logic Programming*, Springer-Verlag, Berlin, 1987.

[11] Michie, D., Memo functions and machine learning, *Nature*, 218, 1968.

[12] Sadri, F. and Kowalski, R., Variants of the event calculus, *Proc. of ICLP*, MIT Press, 1995.

[13] Schank, R., *Conceptual Information Processing*, Elsevier North-Holland, 1975.

[14] Shepherdson, J., Negation as failure, *J. of Logic Programming*, 1, 1984.

[15] Shepherdson, J., Negation as failure, completion and stratification in Gabbay D.,M. et al. (Eds), *Handbook of Logic in AI and Logic Programming, Volume 5, Logic Programming*, Oxford, 1997.

[16] Sripada, S., Rosser, B., Bedford, J., and Kowalski, R., Temporal database technology for air traffic flow management, *Proc. 1st Int. Conf. on Applications of Databases, ADB-94, LNCS, 819*, Springer, 1994.

[17] Stirling, L. and Shapiro, E., *The Art of PROLOG*, 2nd Edition, MIT Press, 1994.

[18] Thomas, R. and Sandhu, R., Discretionary access control in objected-oriented databases : Issues and research directions, *Proc. 16th National Computer Security Conference*, 1993.

Specifying and Computing Hierarchies of Temporal Authorizations

E. Bertino, P.A. Bonatti*, E.Ferrari

Computer Science Department
University of Milano
e-mail:{bertino,ferrarie}@dsi.unimi.it
bonatti@crema.unimi.it

M.L.Sapino

Computer Science Department
University of Torino
e-mail:mlsapino@di.unito.it

Abstract

We present a powerful authorization mechanism which supports: (1) positive and negative temporal authorizations; (2) user-defined temporal rules, by which authorizations can be derived; (3) the hierarchical organization of subjects and objects, with an associated mechanism for authorization inheritance. The resulting model is very flexible in terms of the protection requirements it can represent. Such flexibility requires a non trivial underlying formal model. In particular, when inheritance and derivation rules are used simultaneously, there is need for conditions ensuring that the authorization base is free from ambiguities. For this purpose, we introduce a notion of safeness, *and prove that it guarantees the absence of ambiguities and inconsistencies in the specification. Moreover, we define an efficient algorithm for computing authorizations from safe specifications.*

1 Introduction

In many application domains, the need of restricting access permissions to specific time intervals or periods arises naturally. Authorizations often need to be tailored to the pattern of users' activities; as an example, consider part-time staff that should be authorized for accesses only on working days between 9 a.m. and 1 p.m. Temporal authorizations are also crucial for optimizing resource usage. For instance, execution of resource-expensive programs might be restricted to specific time periods.

Another crucial requirement is the possibility of expressing the relationships that usually exist among authorization subjects and objects. In most application domains, subjects and objects are *hierarchically* organized. The semantics of these hierarchies depends on the considered domain. For instance, the object hierarchy usually defines the composition of an object in terms of other objects (e.g., a directory immediately precedes in the hierarchy the files it contains), whereas the subject hierarchy usually reflects the relative position of subjects within the organization.

In this paper, we present an authorization model that provides integrated support for all the aforementioned features. In the model, *temporal* authorizations can be specified: positive authorizations, for granting a privilege; negative authorizations, for explicitly denying a privilege. A temporal authorization is an authorization that holds for specific periods of time. Subjects and objects are organized into hierarchies, supporting a more adequate representation of their semantics. Authorizations automatically propagate along these hierarchies in that a subject may inherit authorizations specified for more general subjects and objects, *unless* different specific authorizations are explicitly provided. Through inheritance, many protection requirements can be expressed concisely. For instance, a user may authorize all the employees to access the files of a given directory with only one authorization, having as subject the class "employee" and as object the directory. Such an authorization automatically propagates to all the employees and all the files in the directory. Furthermore, owners can always specify positive and negative exceptions to such general authorizations, by issuing authorizations at the lowest levels in the hierarchies. Thus, the resulting model provides a high degree of flexibility coupled with the possibility of enforcing stricter controls on crucial data items.

Our model supports also temporal derivation rules, by which many protection requirements can be concisely and clearly specified. For example, it is possible to state that two users, working on the same project, must receive the same authorizations on certain types of objects, or that a user can access an object in cer-

*This author's work was done while he was at the Department of Computer Science of the University of Torino.

tain periods, provided that nobody else is allowed to access the same object in the same periods.

The work reported in this paper is based on the model presented in [1], that supports both temporal authorizations and derivation rules. The current paper extends [1] with the possibility of hierarchically organizing authorization subjects and objects, and with the related inheritance mechanism. This extension is particularly important when dealing with the interoperability of heterogeneous, distributed systems. In this framework, different local security policies have to be represented and integrated at a global level, to guarantee interoperability and to detect possible inconsistencies and security breaches [8]. Such global representation and integration involves (among other issues) establishing mappings among different object types and subjects. Such mappings, as when dealing with integration of heterogeneous data models, require the notion of hierarchy for both the object types and the subject types. We have therefore extended our previous model with hierarchies in view of supporting temporal authorizations in heterogeneous, distributed systems.

The introduction of hierarchies for authorization objects and subjects raises several theoretical and performance issues. When inheritance and authorization rules are used simultaneously, subtle ambiguities may easily arise, even in very simple specifications.

Example 1.1 Consider the following rules: (R_1) if subject s has write permission on object o, then s has also read permission on o; (R_2) if s does not have read permission on object o, then s must not have write permission on o, either. Assume also that s' is the head of the administration department, and that an authorization A_1 stating that s' can write all documents of his/her department therefore exists. Now consider a classified administration document o'. The security administrator tries to protect this document through a general authorization A_2 stating that no user can read o'. Unfortunately, the resulting security specification is ambiguous. In fact, there are two mutually incompatible possibilities:

1. an authorization A_1', stating that s' can write o', is inherited from A_1, because o' is an instance of the class of documents of the administration department. By R_1, it follows that s' can read o'. Such inferred authorization blocks inheritance from A_2; that is, it is *not* derived that s' cannot read o';

2. an authorization A_2', stating that s' cannot read o', is inherited from A_2, because s' is an instance of the class of all users. By R_2, it follows that s' cannot write o'. This blocks inheritance from A_1; that is, A_1' cannot be derived, and s' cannot write o'.

Clearly, it would be hard to detect such ambiguities in specifications of realistic size, without the help of some automated tool. This task is further complicated by the fact that authorizations can be independently formulated by different users, and, generally, they are not aware of every single authorization and rule in the system. The standard solution to this problem (adopted in [1]) consists in requiring the rules to satisfy so-called *stratifiability* conditions. Unfortunately, stratifiability does not solve the problem; in Example 1.1 above, rules R_1 and R_2 can be encoded in a specification which is stratified, according to [1]. Technically speaking, the problem is that inheritance introduces "hidden rules" that make authorization bases *always* equivalent to *non-stratified* logic programs [4]. Such programs, in general, may be ambiguous (in the above sense) or inconsistent. Moreover no general, efficient way of computing their models is known (the problem is NP-hard). Thus, dealing simultaneously with rules and inheritance requires the development of new techniques.

For this purpose, in this paper, we introduce a notion of *safeness*, and prove that it guarantees the absence of ambiguities and inconsistencies in the specification. Moreover, we define an efficient algorithm for computing authorizations from safe specifications, by dynamically refining a sort of stratification during the computation.

The introduction of temporal features in access control on one side, and the use of logical formalisms for specifying authorizations, on the other side, have been addressed by other research efforts. In particular, the Kerberos [9] system provides a notion of *ticket*, with an associated validity time. The purpose of temporal tickets is very different from our access control model. The former record only the fact that a client has been authenticated by the authentication server; they cannot be used to grant access to specific documents or resources managed by the server.

From the side of logical formalisms for security specifications, Woo and Lam in [11] propose a very general formalism for expressing authorization rules. Their language does not support temporal constraints explicitly, but it has almost the same expressive power as first order logic; this makes it possible to model temporal constraints. The main drawback is that the trade-off between expressiveness and efficiency seems to be strongly unbalanced in their approach. Jajodia

et al. in [6] propose a logic language for expressing authorization rules. Their language can express most of the policies definable through the access control mechanisms proposed so far in the literature; however, temporal authorizations cannot be expressed. The development of flexible authorization models has been addressed in other papers [2, 10]. However none of these proposals support a complete set of features as our model, nor they provide a formal foundation to their models.

The remainder of this paper is organized as follows. Section 2 introduces the formalism we use to represent periodic time. Section 3 illustrates the basic components of the authorization model. Section 4 gives their formal semantic. Section 5 introduces the notion of *safe* authorization base and gives a mechanism to check whether an authorization base satisfies this condition. Section 6 provides an algorithm to efficiently compute the set of authorizations derivable from a safe authorization base. Section 7 concludes the paper and outlines future work. Formal proofs are reported in Appendix A.

2 Periodic Expressions

A symbolic (user friendly) formalism to represent periodic time is provided. The formalism consists of a pair ⟨[begin,end], P⟩ where P is a *periodic expression* denoting an infinite set of time intervals, and [begin,end] denotes the lower and upper bounds that are imposed on time intervals in P.

The formalism for periodic expressions is essentially the one proposed in [7], based on the notion of *calendars*. A calendar is defined as a countable set of contiguous intervals,[1] numbered by integers called *index* of the intervals.

A subcalendar relationship can be established between calendars. Given two calendars C_1 and C_2, we say that C_1 is a subcalendar of C_2, (written $C_1 \sqsubseteq C_2$), if each interval of C_2 is exactly covered by a finite number of intervals of C_1. New calendars can be dynamically constructed from the existing ones, by means of a function *generate*() (cf. [1] for a formal definition), a *reference time instant*, and a *basic calendar* (the tick of the system), denoted by $\perp$. In the following, we postulate the existence of a set of calendars containing at least *Hours*, *Days*, *Weeks*, *Months*, and *Years*, where *Hours* is the calendar with finest granularity.

Calendars can be combined to represent more general sets of periodic intervals, not necessarily contiguous, like, for instance, the set of *Mondays* or the set of *The third hour of the first day of each month.* Complex sets of periodic intervals, like the ones above, are represented by means of *periodic expressions*, formally defined as follows.

Definition 2.1 [1] Given calendars $C_d, C_1, \ldots, C_n$, a *periodic expression* is defined as $\mathtt{P} = \sum_{i=1}^{n} O_i.C_i \rhd r.C_d$, where $O_1 = all$, $O_i \in 2^{\mathbb{N}} \cup \{all\}$ and $C_i \sqsubseteq C_{i-1}$ for $i = 2, \ldots, n$, $C_d \sqsubseteq C_n$, and $r \in \mathbb{N}$.

The left-hand side of $\rhd$ identifies the set of starting points of the intervals, whereas the right-hand side specifies the duration. For example, $all.Years + \{3,7\}.Months \rhd 2.Months$ represents the set of intervals starting at the same instant as the third and seventh month of every year, and having a duration of 2 months. In practice, O_i is omitted when its value is *all*, whereas it is represented by its unique element when it is a singleton. $r.C_d$ is omitted when it is equal to $1.C_n$.

The infinite set of time intervals corresponding to a periodic expression P is denoted by $\Pi(\mathtt{P})$. Function $\Pi()$ is formally defined as follows.

Definition 2.2 [1] Let $\mathtt{P} = \sum_{i=1}^{n} O_i.C_i \rhd r.C_d$ be a periodic expression, then $\Pi(\mathtt{P})$ is a set of time intervals whose common duration is $r \cdot C_d$, and whose set S of starting points is computed as follows:

- if n=1, S contains all the starting points of the intervals of calendar C_1;
- if $n > 1$, and $O_n = \{n_1, \ldots, n_k\}$, then S contains the starting points of the $n_1^{th}, \ldots, n_k^{th}$ intervals (all intervals if $O_n = all$) of calendar C_n included in each interval of $\Pi(\sum_{i=1}^{n-1} O_i.C_i \rhd 1.C_{n-1})$.

For simplicity, in this paper the bounds begin and end will be denoted by *date expressions* of the form mm/dd/yy:hh, with the obvious intended meaning; end can also be ∞. The set of time instants denoted by ⟨[begin,end], P⟩ is formally defined as follows.

Definition 2.3 Let t be a time instant. $t \in Sol([\mathtt{begin}, \mathtt{end}], \mathtt{P})$ iff $t \in \Pi(\mathtt{P})$ and $\mathtt{t}_b \leq t \leq \mathtt{t}_e$, where $\mathtt{t}_b$, $\mathtt{t}_e$ are the instants denoted by begin and end, respectively.

3 The Hierarchical Temporal Authorization Model (HTAM)

In this section we illustrate the basic components of our hierarchical temporal authorization model (HTAM for short).

[1] Two intervals are contiguous if they can be collapsed into a single one (e.g., [1,2] and [3,4]).

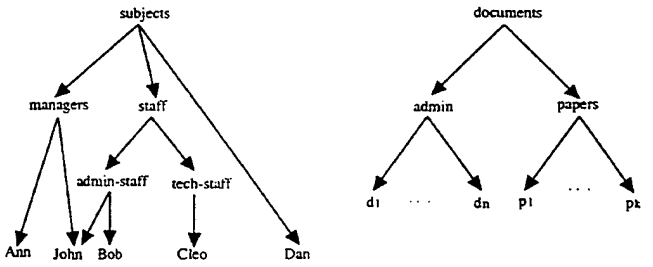


Figure 1: An example of subject and object hierarchies

In the following, S denotes the set of subjects, O is the set of objects, and M is the set of access modes. Members of S are the identifiers with which users connect to the system. We suppose identifiers may refer to single users (e.g., Ann or Bob) or roles (e.g., staff or manager). Subjects and objects are structured into hierarchies. Figure 1 shows an example of subject and object hierarchies.

HTAM supports positive and negative temporal authorizations with the following structure.

Definition 3.1 [1] A *temporal authorization* is a triple ([begin,end],period,auth), where the pair ⟨[begin, end], period⟩ represents a set of time periods, and auth is a 5-tuple (s,o,m,pn,g), with s,g $\in S$, o $\in O$, m $\in M$, and pn $\in \{+,-\}$.

Triple
([begin,end],P,(s,o,m,pn,g)) states that subject s has been authorized (if pn = '+') or denied (if pn = '−') for access mode m on object o by subject g for *each* instant in *Sol*([begin,end],P). For example, the temporal authorization A_1 = ([1/1/96,∞],Mondays,[2] (Matt,o_1,read,+,Bob)), specified by Bob, states that Matt has the authorization to read o_1 each Monday starting from 1/1/96.

We assume that conflicting temporal authorizations may be simultaneously specified. This is a natural assumption both in discretionary frameworks (where different grantors may concurrently grant authorizations on the same objects) and in federated databases (where independently developed local security policies need to be merged to obtain secure interoperation). Conflicts are dealt with according to the denials-take-precedence principle [5], unless one of the conflicting authorizations is more specific than the others with respect to the specified hierarchies. In that case, less specific authorizations are overridden.

[2]Here and in the following we use intuitive names for periodic expressions.

Example 3.1 Consider the hierarchies illustrated in Figure 1. Intuitively, a temporal authorization such as ([1/1/96,∞],⊥,(staff,admin,read,+,Ann)) means that all the staff members can read, starting from 1/1/96, each administration document, *unless explicitly stated otherwise*. So for instance, in the absence of further authorizations, (John,d_3,read,+,Ann) and (Bob,d_7,read,+,Ann) hold, starting from 1/1/96. On the contrary, in the presence of a temporal authorization such as ([1/1/97,∞],⊥,(Bob,d_7,read,-,Ann)), Bob would be allowed to read document d_7 only during 1996. More precisely, the authorization (Bob,d_7,read,+,Ann) would not be inherited after the end of 1996.

Similarly, a positive authorization may block the inheritance of a negative authorization. On the other hand, in case of multiple inheritance, conflicts are resolved in a conservative fashion, according to the denials-take-precedence principle [5].

Example 3.2 Consider the following authorizations:

([1/1/96,∞],⊥,(staff,d_6,write,-,Cleo)),
([1/1/96,∞],⊥,(Bob,admin,write,+,Cleo))},

none is more specific than the other; in this case, the negative authorization is preferred, and only (Bob,d_6,write,-,Cleo) is inherited.

Furthermore, HTAM supports the specification of user-defined *derivation rules*, that makeS it possible to derive further authorizations from those explicitly given. Rule applicability can be constrained to specific time periods. Derivation rules are formally defined as follows.

Definition 3.2 [1] A *derivation rule* is a triple ([begin,end], period, A op F), where the pair ⟨[begin,end],period⟩ represents a set of time periods, A is an authorization, F is a boolean combination of authorizations, and op is one of the operators: whenever, aslongas, upon. The authorization A is called the *head* of the rule; F is called *body*.

Definition 3.3 [1] A *temporal authorization base* (TAB) is a set of temporal authorizations and derivation rules.

Roughly speaking, rule ([begin,end], P, A op F) states that for each instant in *Sol*([begin,end], P), A can be derived provided that F is valid at the

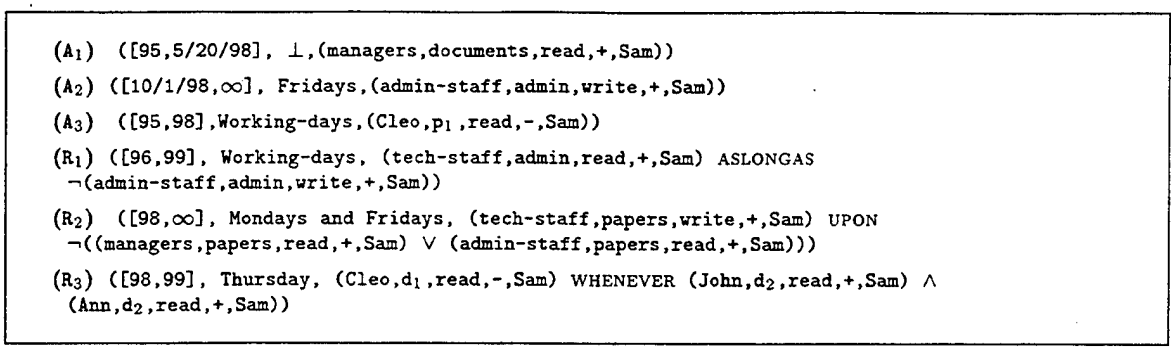


Figure 2: An example of TAB

same instant, or in a certain set of past instants (depending on op). The grantor of A should be the user who specifies the rule.

The following is an intuitive account of the semantics of derivation rules. It will be assumed that all authorizations are granted by the same subject; therefore, the grantor will not be considered in the discussion.

- ([begin,end], P, A WHENEVER F). We can derive A for each instant in Sol([begin, end],P) where F is valid.
- ([begin,end],P, A ASLONGAS F). Authorization A can be derived at instant t in Sol([begin, end],P) if F is valid for each instant in Sol([begin, end],P) from the first one up to t. For instance, rule R_1 in Figure 2 states that tech-staff can read admin each working day in [1/1/96,12/31/99] until the first working day in which admin-staff will be allowed to write admin.
- ([begin,end],P, A UPON F). We can derive A for each instant t in Sol([begin, end],P) such that there exists an instant t' in Sol([begin, end],P) $t' \leq t$ such that F is valid at time t'. For instance, rule R_2 in Figure 2 states that tech-staff can write papers each Monday and Friday starting from the first in 1998 in which both managers and admin-staff are not authorized to read papers.

Example 3.3 Consider the TAB in Figure 2 and the subject and object hierarchies in Figure 1. The following authorizations can be derived:

- (tech-staff,admin,read,+,Sam) for each Working day in [1/1/96,10/2/98], from rule R_1 and authorization A_2. Note that 10/2/98 is the first Friday after 10/1/98.
- (tech-staff,papers,write,+,Sam) for each Monday and Friday from 5/22/98, from rule R_2 and authorization A_1. Note that authorization A_1 implies an authorization for managers to read papers for each day in [1/1/95,5/20/98]. 5/22/98 is the first Friday in [1/1/98,∞] in which neither managers nor staff can read papers.
- (Cleo,d_1,read,-,Sam) for each Thursday from 1/1/98 to 5/20/98 from rule R_3 and authorization A_1. Note that A_1 implies an authorization for Ann and John to read documents for each day from 1/1/95 to 5/20/98 and this authorization implies that Ann and John can read document d_2 for the same days.

4 Formal Semantics of TABs

We start by introducing some notation. The hierarchies of subjects and objects are formalized by two partially ordered sets $(S, \sqsubseteq_S)$ and $(O, \sqsubseteq_O)$, respectively. For example, according to the hierarchies illustrated in Fig. 1, we have (among other relationships) Ann $\sqsubseteq_S$ managers, John $\sqsubseteq_S$ staff, $p_i \sqsubseteq_O$ documents, and $p_i \not\sqsubseteq_O$ admin $(1 \leq i \leq k)$.

Let $\mathcal{A}$ be the set of all the tuples (s,o,m,pn,g). Given an authorization $A = (s, o, m, pn, g)$ we use the notation $s(A)$, $o(A)$, $m(A)$, $pn(A)$, $g(A)$ to denote the corresponding components of A. Let A^+ (resp. A^-) denote the authorization obtained from A by replacing $pn(A)$ with '+' (resp. '−'). For all $A \in \mathcal{A}$, $\bar{A}$ denotes the authorization *complementary to* A, that is, $\bar{A^+} = A^-$ and $\bar{A^-} = A^+$. The relation '$=_g$' defined below is the equality of authorizations modulo

grantors; the set Conflicts(A) captures the set of authorizations conflicting with A:

$$A =_{-g} B \equiv s(A) = s(B) \wedge o(A) = o(B) \wedge m(A) = m(B) \wedge pn(A) = pn(B),$$
$$\mathsf{Conflicts}(A) = \{ B \mid B =_{-g} \bar{A} \}.$$

Note that $B \in \mathsf{Conflicts}(A) \iff A \in \mathsf{Conflicts}(B)$, and $A =_{-g} B \Rightarrow \mathsf{Conflicts}(A) = \mathsf{Conflicts}(B)$. A *conflicting pair* consists of two authorizations A and B such that $A \in \mathsf{Conflicts}(B)$ (equivalently, $B \in \mathsf{Conflicts}(A)$). The subject and object hierarchies induce the following natural partial ordering $\sqsubseteq_{\mathcal{A}}$ over $\mathcal{A}$:

$$A \sqsubseteq_{\mathcal{A}} B \equiv s(A) \sqsubseteq_S s(B) \wedge o(A) \sqsubseteq_O o(B) \wedge m(A) = m(B) \wedge pn(A) = pn(B) \wedge g(A) = g(B).$$

As usual, $A \sqsubset_{\mathcal{A}} B \equiv A \sqsubseteq_{\mathcal{A}} B \wedge A \neq B$. Finally, define the set:

$$\mathsf{Parents}_{\sqsubseteq_{\mathcal{A}}}(A) = \{ B \mid A \sqsubset_{\mathcal{A}} B \text{ and } \nexists C . A \sqsubset_{\mathcal{A}} C \sqsubset_{\mathcal{A}} B \}.$$

The subscripts of $\sqsubseteq_{\mathcal{A}}$ and $\mathsf{Parents}_{\sqsubseteq_{\mathcal{A}}}$ will often be dropped to enhance readability.

Access control is based on the set of *time-stamped authorizations* that can be derived from the given TAB T. A time-stamped authorization is a pair (t, A) where t is a non-negative integer (a "tick" in the basic calendar) and $A \in \mathcal{A}$ (the intuitive meaning is "A holds at time t").

Formalizing derivability involves delicate technicalities due to the presence of *negation as failure*. Rules such as $R = ([t,t], \bot, A \texttt{ whenever } \neg B^-)$ are triggered only if (t, B^-) is *not derivable*, and (t, A) should be inherited only if $(t, \bar{A})$ is *not derivable* (according to the overriding mechanism outlined in the previous section). Consequently, derivable authorizations cannot be defined incrementally through a classical inductive definition; for example, the above rule R may be applicable at a certain stage of the derivation process, but as deduction goes on, (t, B^-) might become derivable, thereby invalidating the conclusion of R. Informally speaking, in order to decide whether R can be applied we should *guess* whether (t, B^-) will ever be derived in the future. Fortunately, a similar form of negation has already been formalized in logic programming, through the *stable model semantics* [3]; here we adopt the same idea.

Intuitively, the definition of derivable authorizations models the following three steps:[3] (i) First, the set of all derivable time-stamped authorizations, I, is guessed; (ii) the consequences of the given TAB T are derived incrementally, by recursively applying all the applicable rules of T and inheritance; the applicability of inheritance and rules with negative literals in the body—like R—is checked using I to "guess the future"; (iii) clearly, the initial guess is correct if it coincides with what can be actually derived with the given rules, i.e. if I equals the set of authorizations derived in step (ii).

Then, the formal definition corresponding to the above steps is just a fixpoint equation $I = \Phi_T^I \uparrow \omega$, where I is the initial guess and $\Phi_T^I \uparrow \omega$ is the set of authorizations obtained in step (ii). In particular, Φ_T^I models a single application (in parallel) of all the applicable rules of T,[4] and the operator $\uparrow \omega$ iterates Φ_T^I.[5]

To formalize Φ_T^I in a compact way, we introduce a notion of *validity relative to the initial guess* I. Recall from the previous section that conflicting authorizations may be simultaneously derivable from T; in this case the denials-take-precedence principle is applied; *validity* captures this idea, and it can be extended to arbitrary boolean combinations of authorizations in negation normal form (NNF).[6] Intuitively, in the following, J stands for the set of authorizations that have been derived at some intermediate step of the derivation (I is the initial guess).

Definition 4.1 Let I and J be sets of time-stamped authorizations, t be a time point, $A \in \mathcal{A}$ and F, G_i be boolean combinations of authorizations from $\mathcal{A}$ in NNF. The *validity* of F in J at time t w.r.t. I, in symbols $J, I \models_t F$, is recursively defined as follows:

1. $J, I \models_t A^+$ if $(t, A^+) \in J$ and for all $B^- \in \mathsf{Conflicts}(A^+)$, $(t, B^-) \notin I$;
2. $J, I \models_t A^-$ if $(t, A^-) \in J$;
3. $J, I \models_t \neg A^+$ if $(t, A^+) \notin I$ or there is $B^- \in \mathsf{Conflicts}(A^+)$ s.t. $(t, B^-) \in J$;
4. $J, I \models_t \neg A^-$ if $(t, A^-) \notin I$;
5. $J, I \models_t G_1 \wedge G_2$ if $J, I \models_t G_1$ and $J, I \models_t G_2$;
6. $J, I \models_t G_1 \vee G_2$ if $J, I \models_t G_1$ or $J, I \models_t G_2$.

Now we can define the *immediate consequence* operator Φ_T^I that applies in parallel all applicable rules and performs all possible inheritance steps. In the following, J plays the role of the set of authorizations

[3] The efficient algorithms introduced in the next section apply only to *safe* TABs and do not match the generate-and-test nature of the following steps.

[4] Note that this operator depends on I, as required by step (ii).

[5] We recall the recursive definition of $\uparrow$: $\Phi \uparrow 0 = \Phi(\emptyset)$; $\Phi \uparrow i + 1 = \Phi(\Phi \uparrow i)$; $\Phi \uparrow \omega = \bigcup_{i \geq 0} \Phi \uparrow i$.

[6] A formula is in negation normal form if negation is applied only to atoms. This is the form of the bodies of TAB rules.

which have already been derived, and $\Phi_T^I(J)$ is the result of one more derivation step.

Definition 4.2

$$\Phi_T^I(J) = \{\,(t,A) \mid (TI,P,A) \in T \text{ and } t \in Sol(TI,P)\,\}$$
$$\cup\ \{\,(t,A) \mid (TI,P,A \text{ whenever } F) \in T,\ t \in Sol(TI,P),\ J,I \models_t F\,\}$$
$$\cup\ \{\,(t,A) \mid (TI,P,A \text{ upon } F) \in T,\ t \in Sol(TI,P) \text{ and } \exists t' \in Sol(TI,P) \text{ such that } t' \le t,\ J,I \models_{t'} F\,\}$$
$$\cup\ \{\,(t,A) \mid (TI,P,A \text{ aslongas } F) \in T,\ t \in Sol(TI,P) \text{ and for all } t' \in Sol(TI,P) \text{ such that } t' \le t,\ J,I \models_{t'} F\,\}$$
$$\cup\ \{\,(t,A) \mid \exists A_1 \in \mathsf{Parents}(A).\ J,I \models_t A_1 \text{ and } \forall A_2 \in \mathsf{Conflicts}(A).\ A_2 \notin I, \text{ and if } A = A^+ \text{ then } \forall A_3 \in \mathsf{Parents}(\mathsf{Conflicts}(A)),\ J,I \models_t \neg A_3\,\}.$$

The first four sets in the right-hand side formalize the semantics of TAB rules; the last set defines the semantics of inheritance. The precedence of negative authorizations over positive authorizations is obtained by inheriting A^+ only when no conflicting authorization $B \in \mathsf{Conflicts}(A^+)$ can possibly be inherited; in turn, this is verified by checking that no parent A_3 of B is valid.

As we have already pointed out, by iterating the above operator we obtain all the facts derivable from T, given the initial guess I.

Proposition 4.1 *The operator Φ_T^I is continuous, and hence i) $\Phi_T^I \uparrow 0 \subseteq \Phi_T^I \uparrow 1 \subseteq \ldots \subseteq \Phi_T^I \uparrow \omega$; ii) the least fixpoint of Φ_T^I is $\Phi_T^I \uparrow \omega$.*

Thus, the initial guess is correct iff I coincides with $\Phi_T^I \uparrow \omega$. So the notion of $\sqsubseteq$-stable model introduced below captures the set of derivable authorizations.

Definition 4.3 A set of time-stamped authorizations I is a *$\sqsubseteq$-stable model* of a TAB T iff $I = \Phi_T^I \uparrow \omega$.

Suppose that T has one $\sqsubseteq$-stable model M; then an authorization A should be enforced at time t iff A is valid in M at time t.

5 Dynamically Stratified TABs

A TAB may have multiple $\sqsubseteq$-stable models, or no $\sqsubseteq$-stable models at all. In this section, we introduce conditions that guarantee that the TAB has exactly one $\sqsubseteq$-stable model (and hence a consistent, non ambiguous semantics).

A *labeled dependency graph* is a graph whose nodes are authorizations from $\mathcal{A}$ and whose edges are labeled with $+$, $-$ or v (accordingly, the edges are called *positive, negative* or *variable*).

Let $DG_T(t)$ be the labeled dependency graph whose set of nodes is $\mathcal{A}$ and whose edges are all and only the triples $\langle A,l,B\rangle$ that satisfy some of the following conditions:

(DG1) For some rule $(TI,P,B\,\langle\,\mathrm{op}\,\rangle\,F) \in T$ such that $t \in Sol(TI,P)$, A occurs in F; moreover, if A occurs in the scope of a negation sign ($\neg$), then $l = -$, otherwise $l = +$;

(DG2) $A \in \mathsf{Conflicts}(A_1^+)$, where A_1^+ occurs in the body F of some rule $(TI,P,B\,\langle\,\mathrm{op}\,\rangle\,F) \in T$ such that $t \in Sol(TI,P)$; moreover, if A_1^+ occurs in the scope of a negation sign ($\neg$), then $l = +$, otherwise $l = -$;

(DG3) $A \in \mathsf{Parents}(B)$ and $l = -$;

(DG4) $A \in \mathsf{Conflicts}(B)$ and $l = v$.

TABs will be required to satisfy the following *safeness conditions*. We shall prove that safeness guarantees the existence of a unique stable model.

Definition 5.1 A TAB T is *safe* if for all $t \ge 0$: *1)* no cycle in $DG(t)$ contains a negative edge; *2)* if A,B is a conflicting pair, then every path from A to B in $DG(t)$ contains a variable edge (A',v,B') such that $A' =_g A$ and $B' =_g B$.

A *stratification* of a set of authorizations $\mathcal{A}' \subseteq \mathcal{A}$ is a mapping $\pi : \mathcal{A}' \to \{1,\ldots,|\mathcal{A}'|\}$. We say that π is *compatible with* a labeled dependency graph DG if: *1)* for all edges $\langle A,l,B\rangle$ in DG, $\pi(A) \le \pi(B)$, and *2)* if $l = -$ then $\pi(A) < \pi(B)$.

In the standard logic programming setting, the existence of a compatible stratification is equivalent to the "safeness" of the program. On the contrary, here safeness is stronger (the second condition is not enforced by the existence of a stratification). Intuitively, the problem is that for every conflicting pair A,B, the inheritability of A depends on the absence of B and viceversa. Fortunately, this cyclic (non-stratifiable) dependency does not affect the nice properties of stratifiable programs (i.e., the existence of a unique stable model which can be computed in polynomial time).

The model of a TAB T which satisfies the safeness conditions can be constructed as follows. Let $\pi_t : \mathcal{A} \to \{1,\ldots,|\mathcal{A}|\}$ be a stratification compatible with $DG_T(t)$. We call π_t a *pre-stratification* of $\mathcal{A}$ at time t. Define $\mathcal{A}_{t,l} = \{\,A \mid \pi_t(A) = l\,\}$. A *refined stratification* of $\mathcal{A}_{t,l}$ w.r.t. an interpretation J is a mapping $\sigma_{t,l} : \mathcal{A}_{t,l} \to \{1,\ldots,|\mathcal{A}_{t,l}|\}$ such that for all $A,B \in \mathcal{A}_{t,l}$:

1. if $\langle A, +, B \rangle$ belongs to $DG_T(t)$ then $\sigma_{t,l}(A) \leq \sigma_{t,l}(B)$;
2. if $\langle A^+, v, B \rangle$ is in $DG_T(t)$ and $\exists B' \in$ Parents(Conflicts(A)). $J, J \models_t B'$ then $\sigma_{t,l}(A) < \sigma_{t,l}(B)$;
3. if $\langle A^-, v, B \rangle$ is in $DG_T(t)$ and $/\exists A' \in$ Parents(Conflicts(B)). $J, J \models_t A'$ then $\sigma_{t,l}(A) < \sigma_{t,l}(B)$.

Note that $\sigma_{t,l}$ is a stratification compatible with a modified version of $DG_T(t)$, where the variable edges over $\mathcal{A}_{t,l}$ have been either eliminated or turned into negative edges. The idea is that the negative cycles involving conflicting pairs can be "broken" dynamically, on the basis of the part of model built so far (represented here by J). Indeed, in the full paper it is shown that for each conflicting pair A, B, either $\sigma_{t,l}(A) < \sigma_{t,l}(B)$ or $\sigma_{t,l}(B) < \sigma_{t,l}(A)$.

The model M is constructed through three nested iterations, over time, pre-stratification level and refined stratification level, respectively. In order to simplify notation, we shall use τ as an abbreviation of an arbitrary triple (t, l, i) of the form $(0,0,0)$ or such that $t \geq 0$, $1 \leq l \leq |\mathcal{A}|$ and $1 \leq i \leq |\mathcal{A}_{t,l}|$. With a slight abuse of notation, the lexicographic ordering of such triples will be denoted by $\leq$, and the immediate predecessor and successor of τ in this ordering will be denoted by $\tau - 1$ and $\tau + 1$, respectively. Each time-stamped authorization (t, A) will be associated to a *layer* $\ell(t, A) = (t, l, i)$, where $l = \pi_t(A)$ and $i = \sigma_{t,l}(A)$. Finally, define by mutual recursion: [7]

$$M = \bigcup_{\tau \geq (0,0,0)} M_\tau, \quad \text{where}$$

$$M_\tau = \emptyset \quad \text{if } \tau = (0,0,0), \text{ otherwise}$$
$$M_\tau = \Phi_{T_\tau}^{M_{\tau-1}} \uparrow \omega(M_{\tau-1}), \quad \text{where}$$

$$\begin{aligned} T_{(t,l,i)} = \{&([t_b, t], P, A \langle \text{op} \rangle F) \mid \\ &([t_b, t_e], P, A \langle \text{op} \rangle F) \in T, \\ &t \in Sol([t_b, t_e], P),\ \sigma_{t,l}(A) = i\} \\ \cup\ \{&([t_b, t], P, A) \mid ([t_b, t_e], P, A) \in T, \\ &t \in Sol([t_b, t_e], P),\ \sigma_{t,l}(A) = i\} \end{aligned}$$

and where $\sigma_{t,l}$ is a refined stratification of $\mathcal{A}_{t,l}$ w.r.t. $M_{(t,l,1)-1}$. Roughly speaking, $T_{(t,l,i)}$ is the set of rules and authorizations that can possibly be used to derive time-stamped authorizations whose layer is (t, l, i). Note that $T_{(t,l,i)}$ depends on $M_{(t,l,1)-1}$ through $\sigma_{t,l}$.

[7] Recall that $\Phi_T^I \uparrow 0(J) = J$ and $\Phi_T^I \uparrow i+1(J) = J \cup \Phi_T^I(\Phi_T^I \uparrow i(J))$.

First, it must be shown that the above construction is well defined, by proving that pre-stratifications π_t and refined stratifications $\sigma_{t,l}$ exist.

Theorem 5.1 *If T satisfies the safeness conditions then π_t and $\sigma_{t,l}$ exist for all $t \geq 0$ and for all l $(1 \leq l \leq |\mathcal{A}|)$.*

Theorem 5.2 *M is the unique $\sqsubseteq$-stable model of T.*

Definition 5.1 above gives a method to test whether a given TAB satisfies the safeness conditions. Such test is performed by checking a set of conditions, for each instant $t \geq 0$. Obviously, this test is feasible in practice only if we can safely stop it at some finite instant t. The particular form our rules allows us to ensure the existence of this finite constant, denoted in the following as `max-time`. `max-time` is determined as $\bar{t}_{max} + \bar{k} \cdot \mathrm{P}_{max}$, where $\bar{t}_{max}$ is the greater instant corresponding to an end date expression in TAB, P_{max} is the least common multiple of all the periodicities of the periodic expressions appearing in TAB, and $\bar{k}$ is the maximum number of ASLONGAS and UPON rules in TAB plus one. The important property is that, after instant `max-time`, the validity of any authorization in TAB becomes periodic. The proof of the existence of `max-time`, adapted from [1], is contained in Appendix A.

6 Access Control

Two different strategies can be used to enforce access control: *run-time derivation* and *materialization*. Under the latter approach, the system permanently maintains all the valid authorizations derivable from a TAB. Access control therefore becomes very efficient; as a drawback, the materialization has to be properly updated every time the TAB is modified.

Like in [1], we adopt the materialization approach. The main reason behind this choice is that in real systems access requests are generally considerably more frequent than requests modifying authorizations and/or rules.

In Appendix A, a natural extension of the results in [1] to our case is presented. In particular, it is proved that there exists an instant t_l which can be used to limit the computation. The model at time t_l can be used to check the validity of any authorization for each instant $t \geq 0$. Given a model M, we call the *compact version* of M, denoted as M^C, the subset of M satisfying the above condition.

Figure 3 presents an algorithm to compute the compact version of the model of a TAB T which satisfies the safeness condition.

Algorithm 5.1
INPUT: A TAB T satisfying the safeness condition
OUTPUT: The compact version M^C of the model M of T
METHOD:

1. Let $\mathtt{t}_{min}$ be the minimum instant corresponding to a begin date expression in T
2. Let $\bar{\mathtt{t}}_{max}$ be the greater instant corresponding to an end date expression in T
3. Let $\mathtt{P}_{max}$ be the lcm of all the periodicities of the periodic expressions in T
4. Let $\bar{k}$ be the maximum number of ASLONGAS and UPON rules in T plus one
5. $\mathtt{max\text{-}time} := \bar{\mathtt{t}}_{max} + \bar{k} \cdot \mathtt{P}_{max}$, $k := 1$, $success := false$, $M^C := \emptyset$
6. Compute π_t, for each t such that $\mathtt{t}_{min} \leq t \leq \bar{\mathtt{t}}_{max} + \mathtt{P}_{max}$
7. **Repeat**
   $k := k + 1$, $current\_max := \bar{\mathtt{t}}_{max} + k \cdot \mathtt{P}_{max}$
   Compute π_t, for each t such that $current\_max - \mathtt{P}_{max} < t \leq current\_max$
   For $t := \mathtt{t}_{min}$ to $current\_max$ **do**
   Let $l^* := max\{l \mid l = \pi_t(\mathtt{A})\}$
   For $i := 1$ to l^* **do**
   Let $T_{t,i} := T(t,i) \cup INH(M^C, t, i) \cup TAB(M^C)$
   Let $L_1, \ldots L_h$ be a stratification of $T_{t,i}$
   For $j := 1$ to h **do**
   Let $T_{t,i,j} := \{ ([t_b, t], P, A\langle\,\mathtt{op}\,\rangle F) \mid ([t_b, t], P, A\langle\,\mathtt{op}\,\rangle F) \in T_{t,i},$
   $(t, A) \in L_j \} \cup \{ ([t_b, t], P, A) \mid ([t_b, t], P, A) \in T_{t,i}, (t, A) \in L_j \}$
   Repeat
   For each $x \in T_{t,i,j}$ **do**
   If $x = ([t_b, t], P, A\langle\,\mathtt{op}\,\rangle F)$ and $\forall B^- \in \mathtt{Conflicts}(A)$ $(t, B^-) \notin M^C$ **then**
   If OP = WHENEVER and $M^C \models_t F$ **then add** (t, A) to M^C
   If OP = ASLONGAS and $\forall t'$ $t_b \leq t' \leq t$ $M^C \models_{t'} F$ **then add** (t, A) to M^C
   If OP = UPON and $\exists t'$ $t_b \leq t' \leq t$ such that $M^C \models_{t'} F$ **then add** (A, t) to M^C
   endif
   If $x = ([t, t], P, A)$ and $\forall B^- \in \mathtt{Conflicts}(A)$ $(t, B^-) \notin M^C$ **then add** (t, A) to M^C
   Until M^C does not change
   endfor
   If $C_{k-1} \stackrel{=}{=} C_k$ **then** $success := true$
   Until *success* or $current\_max > \mathtt{max\text{-}time}$

Figure 3: An algorithm for computing the compact version of the model of a TAB

In Algorithm 5.1, the treatment of the temporal aspects is similar to the one introduced in an analogous algorithm in [1]. The treatment of the derivation rules is based on the model computation given in Section 5. It computes the difference between $M_{(t,l,0)}$ and $M_{(t,l+1,0)}$ (for increasing l and t) by transforming the part of the TAB relevant to time t and level $l+1$ into a TAB without inheritance. For this purpose, for all sets of time-stamped authorizations M, for all time points t and levels l let:

$$
\begin{aligned}
TAB(M) &= \{ ([t,t], \bot, A) \mid (t, A) \in M \}, \\
INH(M,t,l) &= \{ ([t,t], \bot, A \ \mathtt{whenever}\ F) \mid \pi_t(A) = l, \\
&\quad \exists A_1 \in \mathsf{Parents}(A).M \models_t A_1, \\
&\quad \text{if } A = A^+ \text{ then } \forall A_2 \in \mathsf{Parents}(\mathsf{Conflicts}(A)), \\
&\quad M \not\models_t A_2, \ F = \bigwedge_{B \in \mathsf{Conflicts}(A)} \neg B \}, \\
T(t,l) &= \{ ([t_b, t], P, A\langle\,\mathtt{op}\,\rangle F) \mid \\
&\quad ([t_b, t_e], P, A\langle\,\mathtt{op}\,\rangle F) \in T, \\
&\quad t \in Sol([\mathtt{begin}, \mathtt{end}], \mathtt{P}), \ \pi_t(A) = l \} \\
&\quad \cup \{ ([t_b, t], P, A) \mid ([t_b, t_e], P, A) \in T, \\
&\quad\quad t \in Sol([\mathtt{begin}, \mathtt{end}], \mathtt{P}), \ \pi_t(A) = l \}.
\end{aligned}
$$

Intuitively, $TAB(M)$ transforms its argument (that will be the part of the model already computed) into an equivalent TAB; $INH(M,t,l)$ transforms the applicable inheritance rules into explicit whenever rules; $T(t,l)$ is the restriction of T's rules and authorizations to layer l at time t. Finally, for all t, l s.t. $t+l > 0$, let: $T_{t,l} = T(t,l) \cup INH(M_{(t,l,0)-1}, t, l) \cup TAB(M_{(t,l,0)-1})$. The next theorem states that the above transformation yields a well-behaved TAB that captures exactly level l at time t.

Theorem 6.1 *For all t, l s.t. $t + l > 0$, i) $T_{t,l}$ is a stratified TAB; ii) The stable model of $T_{t,l}$ agrees on validity with $M_{(t,l+1,0)-1}$.*

The complete algorithm works as follows. First, it computes π_t for each instant t between $\mathtt{t}_{min}$ and *current_max*, where $\mathtt{t}_{min}$ is the minimum instant corresponding to a begin date expression in TAB and *current_max* is initially set equal to $\overline{\mathtt{t}}_{max}+\mathtt{P}_{max}$ and incremented by $\mathtt{P}_{max}$ at each iteration. Then, for each level l of the stratification and for each instant t between $\mathtt{t}_{min}$ and *current_max*, the algorithm transforms the part of the TAB relevant to time t and level l into a TAB without inheritance (using the method sketched above). According to Theorem 6.1 the resulting TAB is stratified. Thus, we can apply the algorithm developed for such TABs (see [1]) to compute a stratification for level l, as well as the authorizations of level l derivable from the TAB at time t.

Then, the algorithm considers the computed time-stamped authorizations in two contiguous time intervals after $\overline{\mathtt{t}}_{max}$ of length equal to $\mathtt{P}_{max}$ and checks whether these sets are equivalent ($C_{k-1} \stackrel{\rightarrow}{=} C_k$). If so, the behavior is already periodic and the algorithm terminates. If not, and if the considered interval do not exceed `max-time`, it proceeds with another iteration of the **Repeat-until** cycle, generating a larger model using the constant of the previous iteration incremented by $\mathtt{P}_{max}$.

In practice, we expect the algorithm to terminate at the first iteration in most cases. The following theorem states the termination and the correctness of Algorithm 5.1.

Theorem 6.2 *i) Algorithm 5.1 terminates, and ii) an authorization* A *is valid at time* $\bar{t}$ *iff one of the following conditions holds. Let* $\hat{t}$ *be the greatest instant appearing in a time-stamped authorization in* M^C*, and let* t^* *be the minimum instant such that* $\hat{t} \leq t^*$ *and* $t^* = \overline{\mathtt{t}}_{max} + n \cdot \mathtt{P}_{max}$, $n \in \mathbb{N}$.

- $\bar{t} \leq \hat{t}$, *and* $(\bar{t}, \mathtt{A}) \in M^C$;
- $\bar{t} > \hat{t}$, *and there exists an instant* t', $t^* - \mathtt{P}_{max} < t' \leq t^*$ *such that* $t' = \bar{t} - k \cdot \mathtt{P}_{max}$ *and* $(t', \mathtt{A}) \in M^C$.

7 Conclusions

In this paper we have presented a powerful authorization mechanism which provides support for temporal authorizations, user-defined derivation rules, and the hierarchical organization of subjects and objects.

We have given a solution to the problems related to the simultaneous presence of inheritance and authorization rules. Variable edges, safeness and the dynamic stratification mechanism have no counterpart in the literature. Safeness guarantees that the TAB is consistent and unambiguous. Moreover, we have designed an effective algorithm for computing a compact version of the unique model of safe TABs.

Sometimes, in the presence of ambiguities, it may be possible to select one stable model of the TAB, by taking into account additional information, such as the grantors of the conflicting authorizations, or priorities defined over access modes. This will be the subject of future work.

References

[1] E. Bertino, C. Bettini, E. Ferrari, P. Samarati. An Access Control Model Supporting Periodicity Constraints and Temporal Reasoning. *ACM Trans. on Database Systems*, 23(3):231–285, September 1998.

[2] E. Bertino, S. Jajodia, P. Samarati. Supporting Multiple Access Control Policies in Database Systems. In *Proc. of the IEEE Symposium on Research in Security and Privacy*, Oakland (CA), 1996.

[3] M. Gelfond, V. Lifschitz. The Stable Model Semantics for Logic Programming. In *Proc. Fifth International Conference and Symposium on Logic Programming*, pp. 1070–1080, Seattle, 1988.

[4] T.C. Przymusinski. On the Declarative Semantics of Deductive Databases and Logic Programs. J. Minker editor, *Foundations of Deductive Databases and Logic Programming*, Morgan Kaufmann Publishers, 1987.

[5] S. Jajodia, P. Samarati, V.S. Subrahmanian, E. Bertino. A Unified Framework for Enforcing Multiple Access Control Policies. In *Proc. of the 1997 ACM SIGMOD Int'l Conf. on Management of Data*, Tucson (AZ) May 1997.

[6] S. Jajodia, P. Samarati, V.S. Subrahmanian. A Logical Language for Expressing Authorizations. In *Proc. IEEE Symposium on Security and Privacy*, Oakland (CA), pp. 31–42, May 1997.

[7] M. Niezette, J. Stevenne. An Efficient Symbolic Representation of Periodic Time. In *First International Conf. on Information and Knowledge Management*, Baltimore (MD), 1992.

[8] L. Gong, X. Qian. Computational Issues in Secure Interoperation. *IEEE Trans. on Software Engineering*, 22(1):43–52, 1996.

[9] J. G. Steiner, C. Neuman, J. I. Schiller. Kerberos: An authentication service for open network systems. In *USENIX Conference Proceedings*, pages 191–202, Dallas, TX, 1988.

[10] F. Rabitti, E. Bertino, W. Kim, D. Woelk. A Model of Authorization for Next-Generation Database Systems. *ACM Trans. on Database Systems*, 16(1):88–131, March 1991.

[11] T.Y.C. Woo, S.S. Lam. Authorizations in distributed systems: A new approach. *Journal of Computer Security*, 2(2 & 3):107–136, 1993.

A Main Proofs

Lemma A.1 *Let T be a safe TAB and let M be its model. Let $C_i = \{(t, A) \mid (t, A) \in M$ and $\bar{\mathtt{t}}_{max} + (i-1) \cdot \mathtt{P}_{max} < t \leq \bar{\mathtt{t}}_{max} + i \cdot \mathtt{P}_{max}\}$. Given C_i and C_j, we say that C_i and C_j are shift-equivalent (written $C_i \rightleftarrows C_j$) if for each pair (t,A) in C_i a corresponding one (with the same authorization) can be found in C_j by "shifting" the temporal instant t by the value $(j - i) \cdot \mathtt{P}_{max}$, and vice versa. Thus, here exists an integer $\bar{k}$ such that $C_k \rightleftarrows C_{\bar{k}}$ for each $k > \bar{k}$. The integer $\bar{k}$ is limited by one plus the number of* ASLONGAS *and* UPON *rules in T having an unbound associated interval.*

Proof. To prove the thesis it is sufficient to consider the derivation of pair (t,A) at instants greater than $\bar{\mathtt{t}}_{max}$. Since any derivation of a pair (t,A) with $t > \bar{\mathtt{t}}_{max}$ is possible only through a rule or an authorization with an unbound associated interval (that is, an interval of the form [begin,∞]), we only consider authorizations and rules in T with an unbound associated interval. The proof is articulated in three main steps:

1. $C_i \rightleftarrows C_{i+1}$ implies that there exists a rule in T and two instants t and t' in the i-th period with $t < t'$, such that (i) either the rule operator is ASLONGAS and the rule fires at t and not at t', or (ii) the operator is UPON and the rule does not fire at time t, while it fires at time t'. Intuitively, this statement says that, in the i-th period, either an ASLONGAS rule "expires" or an UPON rule "triggers". To prove this statement we first note that if only explicit authorizations and WHENEVER rules are present in TAB, then $C_i \rightleftarrows C_{i+1}$.

   Hence, $C_i \not\rightleftarrows C_{i+1}$ implies the existence of an ASLONGAS rule R_a or of an UPON rule R_u, and of an authorization A such that the rule derive (t,A) and not (t+$\mathtt{P}_{max}$,A), or viceversa. Consider first the ASLONGAS case with R_a = ([begin, ∞], P, A ASLONGAS F), and let t' with $t < t' \leq t + \mathtt{P}_{max}$ be the first instant in Sol([begin,∞],P) for which rule R_a cannot derive (t',A). Thus, there exists a conjunct L in F such that either *i)* $L = \mathtt{A}'$ and (t',A') $\notin M$ or *ii)* $L = \neg\mathtt{A}'$ and (t',A') $\in M$. t' could be in the i-th period, in which case the above statement would be proved, or in the $(i+1)$-th period. In this last case, since it is the first instant, we have that either *i)* $L = \mathtt{A}'$ ($t' - \mathtt{P}_{max}$,A') $\in M$ and (t',A') $\notin M$ or *ii)* $L = \neg\mathtt{A}'$ ($t' - \mathtt{P}_{max}$,A') $\notin M$ and (t',A') $\in M$. Then, we can recursively apply the same reasoning for L as for $(t, \mathtt{A})$. It is easily seen that, eventually, either we find an ASLONGAS rule that "expires" in the i-th period, or we find a conjunct derived by an UPON rule.

   A similar reasoning can be done for the UPON rule case, i.e., we consider the first instant t' in the i-th period for which the rule R_u *can* derive (t', A). Eventually, either we find an UPON rule that "triggers" in the i-th period, or we find a conjunct derived by an ASLONGAS rule. Hence, the statement in item 1 holds.

2. Given two pairs (C_i, C_{i+1}) and (C_k, C_{k+1}) such that $C_i \not\rightleftarrows C_{i+1}$ and $C_k \not\rightleftarrows C_{k+1}$, the rules expiring (triggering, resp.) in the i-th period and in the k-th period, as stated above, are different ASLONGAS (UPON resp.) derivation rules in TAB. Indeed, consider the case of ASLONGAS. We assume, without loss of generality, that $i < k$. Let R_a = ([begin, ∞], P, A ASLONGAS F) be one of the ASLONGAS rules expiring in the i-th period, and let $t' \in Sol$([begin,∞],P) be the first instant in the i-th period, such that R_a cannot be fired to derive (t',A). Hence, at least one of the conjunct L in F is such that either *i)* $L = \mathtt{A}'$ and (t',A') $\notin M$ or *ii)* $L = \neg\mathtt{A}'$ and (t',A') $\in M$. It is easily seen from the semantics of ASLONGAS that this rule cannot be used anymore to derive authorization A, that is, there cannot exist an instant $t'' > t'$ such that (t',A) $\in M$ and (t',A) is derived from R_a. Suppose, by contradiction, that the rule R_a can be used to derive (t'',A), with t'' in the $(k+1)$-th period. Let us consider the conjunct L in F considered above. Suppose that *i* holds (the proof for *ii)* is analogous) that is, L = A'. Thus, (t^*,A') $\in M$ $\forall t^* \in Sol$([begin,∞],P), $t^* \leq t''$. Thus, (t',A') $\in M$, which is clearly a contradiction. The arguments for UPON are similar and we omit them. This proof immediately leads to the result that the maximum number of 'different" C_is is limited by one plus the maximum number of ASLONGAS and UPON rules in TAB. In-

deed, each rule can be responsible for (at most) one relation $C_i \not\supseteq C_{i+1}$.

3. $C_i \stackrel{\rightarrow}{=} C_{i+1}$ implies $C_i \stackrel{\rightarrow}{=} C_j$ for each $j > i$. To prove this statement it is sufficient to show that $C_i \stackrel{\rightarrow}{=} C_{i+1}$ implies $C_{i+1} \stackrel{\rightarrow}{=} C_{i+2}$. Suppose by contradiction that $C_i \stackrel{\rightarrow}{=} C_{i+1}$ but $C_{i+1} \stackrel{\rightarrow}{\neq} C_{i+2}$. By item 1 above, this means that there exists a rule in TAB and two instants t and t' in the $(i+1)$-th period with $t < t'$, such that (i) either the rule operator is ASLONGAS and the rule fires at t while it does not fire at t', or (ii) the operator is UPON and the rule does not fire at t, while it fires at t'. Consider case (i), and let $(t, \mathtt{A})$ be the pair derived by the rule. Hence, $(t, \mathtt{A}) \in C_{i+1}$ and $(t', \mathtt{A}) \notin C_{i+1}$. However, since $C_i \stackrel{\rightarrow}{=} C_{i+1}$, then $(t' - \mathtt{P}_{max}, \mathtt{A}) \notin C_i$. This means that rule expired in the i-th period and this contradicts the fact that the same rule can derive $(t, \mathtt{A})$ in the $(i+1)$-th period. Consider case (ii), and let $(t', \mathtt{A})$ be the pair derived by the rule. Hence, $(t', \mathtt{A}) \in C_{i+1}$ and $(t, \mathtt{A}) \notin C_{i+1}$. However, since $C_i \stackrel{\rightarrow}{=} C_{i+1}$, then $(t' - \mathtt{P}_{max}, \mathtt{A}) \in C_i$. This means that rule triggered in the i-th period and this contradicts the fact that the same rule cannot derive $(t, \mathtt{A})$ in the $(i+1)$-th period.

From items 2 and 3, we can easily conclude the thesis. □

Proposition A.1 *For each conflicting pair A, B, either $\sigma_{t,l}(A) < \sigma_{t,l}(B)$ or $\sigma_{t,l}(B) < \sigma_{t,l}(A)$.*

Proof For all conflicting pairs A, B, both (A, v, B) and (B, v, A) belong to $DG(t)$. First suppose that A is positive; if $\exists B' \in \mathsf{Parents}(\mathsf{Conflicts}(A)).\, J, J \models_t B'$ then $\sigma(A) < \sigma(B)$ by condition 2 in the definition of refined stratification, otherwise $\sigma(B) < \sigma(A)$ by condition 3. The case where A is negative is symmetric. □

Proof of Theorem 5.1

Claim. For all graphs G with vertices V there exists a function $\sigma : V \to \{1, \ldots, |V|\}$ such that:

1. if there is a path from A to B in G then $\sigma(A) \leq \sigma(B)$;
2. if $\sigma(A) = \sigma(B)$, then A and B belong to the same strongly connected component of G.

To prove the claim, let $G_1, \ldots, G_n$ be the strongly connected components of G, and define $G_i \prec G_j$ iff $i \neq j$ and there is a path in G from a vertex of G_i to a vertex of G_j. Note that $\prec$ is a strict partial order; it is well known that every such order can be extended to a strict total order $\prec^\circ$ that preserves $\prec$. Therefore, there exists a permutation $i_1, \ldots, i_n$ of $1, \ldots, n$ such that $G_{i_1} \prec^\circ G_{i_2} \prec^\circ \ldots \prec^\circ G_{i_n}$. Now, for all G_{i_k} and all vertices A in G_{i_k}, let $\sigma(A) = k$. Clearly, σ satisfies the above two conditions. This concludes the proof of the claim.

Now let t and l be arbitrary integers satisfying the assumptions. Consider the function σ corresponding to $DG(t)$. The first condition of compatibility with $DG(t)$ (namely, for all edges $\langle A, l, B \rangle$ in DG, $\sigma(A) \leq \sigma(B)$) follows immediately from point 1 of the Claim. Suppose that the second compatibility condition (if $l = -$ then $\sigma(A) < \sigma(B)$) is violated for some edge $(A, -, B)$; then $\sigma(A) = \sigma(B)$ and hence, by point 2 of the Claim, A and B belong to the same strongly connected component. Therefore, there must be a path from B to A which can be extended to a negative cycle by means of $(A, -, B)$, contradicting the safeness of T. This proves that σ is compatible with $DG(t)$, and hence the existence of π_t.

We are left to show that $\sigma_{t,l}$ exists. Let $DG_{t,l}$ be the subgraph of $DG(t)$ covering all and only the vertices in $\mathcal{A}_{t,l}$. The function π_t maps all of these vertices onto the same value l; therefore, since T is safe, $DG_{t,l}$ contains no negative edges. Obtain a graph DG from $DG_{t,l}$ by removing all and only the variable edges (A, v, B) that satisfy one of the following conditions

(a) $A = A^+$ and $\not\exists B' \in \mathsf{Parents}(\mathsf{Conflicts}(A))$ s.t. $M_{(t,l,1)-1}, M_{(t,l,1)-1} \models_t B'$;

(b) $A = A^-$ and $\exists A' \in \mathsf{Parents}(\mathsf{Conflicts}(B))$ s.t. $M_{(t,l,1)-1}, M_{(t,l,1)-1} \models_t A'$.

Note that if (a) (resp. (b)) holds for some edge (A, v, B), then (b) (resp. (a)) cannot hold for (B, v, A), nor for any (B'', v, A'') with $A'' =_{-g} A$ and $B'' =_{-g} B$. It follows that DG cannot contain any pair of edges (A, v, B) and (B', v, A') such that $A' =_{-g} A$ and $B' =_{-g} B$.

Let σ be a function satisfying the conditions of the Claim w.r.t. DG. Every positive edge $(A, +, B)$ with $A, B \in \mathcal{A}_{t,l}$ belongs to DG by construction; therefore, point 1 of the Claim implies that σ satisfies the first condition of the definition of refined stratification w.r.t. $M_{(t,l,1)-1}$. Secondly, suppose that the second of such conditions is violated for some edge (A, v, B); that is, $A = A^+$,

$$\exists B' \in \mathsf{Parents}(\mathsf{Conflicts}(A)).\, M_{(t,l,1)-1}, M_{(t,l,1)-1} \models_t B', \quad (1)$$

$$\sigma(A) \not< \sigma(B). \quad (2)$$

By (1), (A, v, B) is not eliminated and belongs to DG; then, by point 1 of the Claim, $\sigma(A) \leq \sigma(B)$; consequently, by (2), $\sigma(A) = \sigma(B)$. This implies, by point 2 of the Claim, that DG contains a path from B to A. By the second safeness condition, such path must contain an edge (B', v, A') such that $B' =_g B$ and $A' =_g A$. But this is absurd, because as we have already pointed out, DG should contain at most one of (A, v, B) and (B', v, A'). Therefore, σ must satisfy the second condition in the definition of refined stratification. Symmetrically, if the third such condition were violated by σ then a contradiction would follow. This proves that σ is a refined stratification on $\mathcal{A}_{t,l}$ w.r.t. $M_{(t,l,1)-1}$. □

Lemma A.2 $M \subseteq \Phi_T^M \uparrow \alpha$.

Proof The following claims will be needed. They follow easily from the definition of Φ_T^I and (for Claim 1) from the assumption that T is safe:

Claim 1. We say that two interpretations I and J *agree until* τ if for all (t, A) with $\ell(t, A) \leq \tau$,

$$(t, A) \in I \iff (t, A) \in J.$$

If I and J agree until $\tau - 1$ then $\Phi_{T_\tau}^I = \Phi_{T_\tau}^J$.

Claim 2. Restricting T to T_τ restricts the output of Φ_T^I, that is, for all triples τ and all interpretations I, J,

$$\Phi_{T_\tau}^I(J) \subseteq \Phi_T^I(J).$$

Now, in order to prove the theorem, it suffices to show that for all $\tau \geq (0, 0, 0)$,

$$M_\tau \subseteq \Phi_T^M \uparrow \omega. \qquad (3)$$

By induction on τ. The base case is trivial since $M_{(0,0,0)} = \emptyset$. Induction step: suppose that (3) holds for all $\tau' < \tau$ but not for τ $(\tau > (0, 0, 0))$, and let j be the least integer such that $\Phi_{T_\tau}^{M_{\tau-1}} \uparrow j(M_{\tau-1}) \not\subseteq \Phi_T^M \uparrow \omega$. It must be $j > 0$, because $\Phi_{T_\tau}^{M_{\tau-1}} \uparrow 0(M_{\tau-1}) = M_{\tau-1} \subseteq \Phi_T^M \uparrow \omega$ (by induction hypothesis). But then

$\Phi_{T_\tau}^{M_{\tau-1}} \uparrow j(M_{\tau-1}) =$

$= M_{\tau-1} \cup \Phi_{T_\tau}^{M_{\tau-1}}(\Phi_{T_\tau}^{M_{\tau-1}} \uparrow j-1(M_{\tau-1}))$ by def.

$= M_{\tau-1} \cup \Phi_{T_\tau}^{M}(\Phi_{T_\tau}^{M_{\tau-1}} \uparrow j-1(M_{\tau-1}))$ by Claim 1, note that M and $M_{\tau-1}$ agree until $\tau - 1$ by construction.

$\subseteq M_{\tau-1} \cup \Phi_T^M(\Phi_{T_\tau}^{M_{\tau-1}} \uparrow j-1(M_{\tau-1}))$ by Claim 2

$\subseteq M_{\tau-1} \cup \Phi_T^M(\Phi_T^M \uparrow \omega)$ by minimality of j and monotonicity of Φ_T^M

$= M_{\tau-1} \cup \Phi_T^M \uparrow \omega = \Phi_T^M \uparrow \omega$ by Prop. 4.1 and induction hypothesis.

A contradiction. □

Proposition A.2 *Given a stratified TAB T, consider the iterative construction of the model M. Any time, in the construction, the validity of a negated authorization $\neg A^+$, at a time instant t, is checked with respect to two interpretations $\Phi_{T_\tau}^{M_{\tau-1}} \uparrow k$ and $M_{\tau-1}$, it is the case that $\ell(t, A^+) < \tau$.*

Proof

It is immediate from the definition of the iterative process. □

Lemma A.3 *If T is a stratified TAB, and, for some $k < \omega$, $M_\tau, M_{\tau-1} \models_t L$, for any authorization literal L, then for any $\tau" > \tau$, it holds that $M_{\tau"}, M_{\tau"-1} \models_t L$.*

Proof The lemma follows from the definition of stratification, the monotonicity of the sequence $M_{\tau_0}, M_{\tau_1}, \ldots, M$, and Proposition A.2.

□

Corollary A.1 *If T is a stratified TAB, and $M_\tau, M_{\tau-1} \models_t F$, where F is any conjunction of authorization literals, then for any $\tau" > \tau$, for any $k < \omega$, it holds that $\Phi_{T_{\tau"}}^{M_{\tau"-1}} \uparrow k(M_{\tau"-1}), M_{\tau"-1} \models_t F$.*

Lemma A.4 $\Phi_T^M \uparrow \alpha \subseteq M$.

Proof By induction on α. *Base:* $\alpha = 0$. $\Phi_T^M \uparrow \alpha = \emptyset \subseteq M$. *Inductive hypothesis:* $(t', B) \in \Phi_T^M \uparrow \alpha - 1 \Rightarrow (t', B) \in M$, for any authorization B, we prove that $(t, A) \in \Phi_T^M \uparrow \alpha \Rightarrow (t, A) \in M$, for any authorization A. Let $\tau = \ell(t, A)$ in the considered dynamic stratification of T. By cases on the rule applied to derive (t, A) in $\Phi_T^M \uparrow \alpha$.

rule A $(TI, P, A) \in T$, and $t \in Sol(TI, P)$. $(TI, P, A) \in T_\tau$ (definition of stratification), and then $(t, A) \in \Phi_{T_\tau}^{M_{\tau-1}} \uparrow 1(M_{\tau-1})$ (by definition of the operator Φ), therefore $(t, A) \in M_\tau \subseteq M$.

rule B $(TI, P, A$ whenever $F) \in T$, $t \in Sol(TI, P)$, and $\Phi_T^M \uparrow \alpha - 1, M \models_t F$.

Let $F = \neg B_1, \ldots, \neg B_n, C_1, \ldots, C_m$.

We prove that there exists $k \leq \omega$ such that

$\Phi_{T_\tau}^{M_{\tau-1}} \uparrow k(M_{\tau-1}), M_{\tau-1} \models_t F$.

In particular, we prove that there exists $k \leq \omega$ such that

$\Phi_{T_\tau}^{M_{\tau-1}} \uparrow k(M_{\tau-1}), M_{\tau-1} \models_t \neg B_i$, for all $1 \leq i \leq n$, and

$\Phi_{T_\tau}^{M_{\tau-1}} \uparrow k(M_{\tau-1}), M_{\tau-1} \models_t C_j$, for all $1 \leq j \leq m$.

Consider any B_i.

$\Phi_T^M \uparrow \alpha - 1, M \models_t \neg B_i$, from the hypothesis.

- If $B_i = B_i^+$, then either (i) $(t, B_i) \notin M$, or (ii) $\exists D^- \in \mathsf{Conflicts}(B_i)$ such that $(t, D^-) \in \Phi_T^M \uparrow \alpha - 1$.

  In the first case, $(t, B_i) \notin M_{\tau-1}$ (since $M_{\tau-1} \subseteq M$, by construction).

  In the second case $(t, D^-) \in \Phi_T^M \uparrow \alpha - 1 \Rightarrow (t, D^-) \in M$ (inductive hypothesis) $\Rightarrow (t, D^-) \in M_{\tau'}$, being $\tau' = \ell(t, D^-)$ (from the stratification of T).

  $D^- \in \mathsf{Conflicts}(B_i) \Rightarrow \exists (D^-, +, A) \in DG_T(t)$, and then $\Pi_t(D^-) \leq \Pi_t(A)$ (definition of stratification). Similarly,

  $D^- \in \mathsf{Conflicts}(B_i) \Rightarrow \exists (D^-, v, B_i) \in DG_T(t)$, and $\exists (B_i, v, D^-) \in DG_T(t)$, and then $\Pi_t(D^-) = \Pi_t(B_i)$.

  Being $\Pi_t(B_i) < \Pi_t(A)$ (stratification), we can conclude that $\Pi_t(D^-) < \Pi_t(A)$, and then $\tau' = \ell(t, D^-) < \ell(t, A) = \tau$. Therefore, $(t, D^-) \in M_{\tau-1}$.

  In both cases (i) and (ii), for any $k < \omega$, it holds that $\Phi_{T_\tau}^{M_{\tau-1}} \uparrow k(M_{\tau-1}), M_{\tau-1} \models_t \neg B_i$.

- If $B_i = B_i^-$, it must be $\Phi_T^M \uparrow \alpha - 1, M \models_t \neg B_i$, and then $(t, B_i) \notin M$. As a consequence, $(t, B_i) \notin M_{\tau-1}$, which proves that $\Phi_{T_\tau}^{M_{\tau-1}} \uparrow k(M_{\tau-1}), M_{\tau-1} \models_t \neg B_i$.

Consider any C_j.

- If $C_j = C_j^+$, $\Phi_T^M \uparrow \alpha - 1, M \models_t C_j \Rightarrow (t, C_j) \in \Phi_T^M \uparrow \alpha - 1$, and $\forall D^- \in \mathsf{Conflicts}(C_j), (t, D^-) \notin M$ (definition of validity).

  $(t, C_j) \in \Phi_T^M \uparrow \alpha - 1 \Rightarrow (t, C_j) \in M$ (inductive hypothesis) $\Rightarrow (t, C_j) \in M_{\tau'}$, with $\tau' = \ell(t, C_j), \tau' \leq \tau$ (stratification of T) $\Rightarrow (t, C_j) \in M_\tau$.

  This proves that there exists $k_1 < \omega$ such that $(t, C_j) \in \Phi_{T_\tau}^{M_{\tau-1}} \uparrow k_1(M_{\tau-1})$.

  Besides, $(t, D^-) \notin M \Rightarrow (t, D^-) \notin M_{\tau-1}$, and therefore $\Phi_{T_\tau}^{M_{\tau-1}} \uparrow k_1(M_{\tau-1}), M_{\tau-1} \models_t C_j$.

- If $C_j = C_j^-$, it must be that $(t, C_j) \in \Phi_T^M \uparrow \alpha - 1$.

  Reasoning as in the previous case we can conclude that there exists $k_2 < \omega$ such that $\Phi_{T_\tau}^{M_{\tau-1}} \uparrow k_2(M_{\tau-1}), M_{\tau-1} \models_t C_j$.

  Globally, we can conclude that there exists $k (= max(k_1, k_2))$ such that $\Phi_{T_\tau}^{M_{\tau-1}} \uparrow k(M_{\tau-1}), M_{\tau-1} \models_t F$, and then, by definition of the Φ operator,

  $(t, A) \in \Phi_{T_\tau}^{M_{\tau-1}} \uparrow k + 1(M_{\tau-1}) \subseteq M$.

rule C $(TI, P, A \text{ upon } F) \in T$, $t \in Sol(TI, P)$, and $\exists t' \in Sol(TI, P)$ such that $t' \leq t$, and $\Phi_T^M \uparrow \alpha - 1, M \models_{t'} F$.

If $t' = t$, this case reduces to the previous one (rule B).

If $t' < t$, let $\tau' = \ell(t', A)$.

$\tau' < \tau \Rightarrow \Phi_{T_\tau}^{M_{\tau-1}} \uparrow k(M_{\tau-1}), M_{\tau-1} \models'_t F$,

(Corollary A.1) and then $(t, A) \in M_\tau$, by definition of the immediate consequence operator.

rule D $(TI, P, A \text{ aslongas } F) \in T$, $t \in Sol(TI, P)$, and $\forall t' \in Sol(TI, P)$ such that $t' \leq t$, $\Phi_T^M \uparrow \alpha - 1, M \models_{t'} F$.

If $t' = t$ is the only $t' \in Sol(TI, P)$ such that $t' \leq t$, this proof reduces to the case of rule B.

Otherwise, for any $t' < t$, let $\tau' = \ell(t', A)$. $\Phi_{T_{\tau'}}^{M_{\tau'-1}} \uparrow k(M_{\tau'-1}), M_{\tau'-1} \models'_t F \Rightarrow M_{\tau'}, M_{\tau'-1} \models'_t F$.

Since $\tau' < \tau$ it holds that for any $k < \omega$, $\Phi_{T_\tau}^{M_{\tau-1}} \uparrow k(M_{\tau-1}), M_{\tau-1} \models'_t F$, by corollary A.1, and then $(t, A) \in M_\tau$, by definition of the immediate consequence operator.

rule E $\exists A_1 \in \mathsf{Parents}(A)$ such that

- $\Phi_T^M \uparrow (\alpha - 1), M \models_t A_1$.

  We prove that there exists $k < \omega$ such that $\Phi_{T_\tau}^{M_{\tau-1}} \uparrow k(M_{\tau-1}), M_{\tau-1} \models_t A_1$,

  Let $A = A_1^+$.

  $\Phi_T^M \uparrow (\alpha - 1), M \models_t A_1^+ \Rightarrow (t, A_1^+) \in \Phi_T^M \uparrow \alpha - 1$ and $\forall B^- \in \mathsf{Conflicts}(A_1^+)(t, B^-) \notin M$, by definition of validity.

  $(t, A_1^+) \in \Phi_T^M \uparrow \alpha - 1 \Rightarrow (t, A_1^+) \in M$ (inductive hypothesis) $\Rightarrow (t, A_1) \in M_{\tau_1}$, where $\tau_1 = \ell(t, A_1^+)$. Since $A_1 \in \mathsf{Parents}(A) \Rightarrow \tau_1 < \tau$ (by definition of stratification), and then $(t, A_1^+) \in \Phi_{T_\tau}^{M_{\tau-1}} \uparrow 0$.

  Besides, $\forall B^- \in \mathsf{Conflicts}(A_1^+)(t, B^-) \notin M \Rightarrow (t, B^-) \notin M_{\tau-1}$ (monotonicity). Therefore, $\Phi_{T_\tau}^{M_{\tau-1}} \uparrow 0(M_{\tau-1}), M_{\tau-1} \models_t A_1^+$,

If $A = A^-$, the proof is analogous to the first part of the above case.

- $\forall A_2 \in \mathsf{Conflicts}(A), (t, A_2) \notin M$. Therefore $(t, A_2) \notin M_{\tau-1}$ (monotonicity).
- if $A = A^+$, then $\forall A_3 \in \mathsf{Parents}(\mathsf{Conflicts}(A)), \Phi_T^M \uparrow (\alpha - 1), M \models_t \neg A_3$.
  By definition of validity, it must be $(t, A_3) \notin M$, and then $(t, A_3) \notin M_{\tau-1}$ (monotonicity). Therefore, $M_{\tau-1}, M_{\tau-1} \models_t A_3$.

Thus, it is proved that $(t, A) \in M$.

□

Proof of Theorem 5.2 M is a stable model of T by lemmas A.2 and A.4. To prove uniqueness, let N be any stable model of T. Suppose that $N \neq M$ and let $\bar{\tau}$ be the maximal τ such that N and M agree until τ. It suffices to show that for all i, $\Phi_T^N \uparrow i$ and $\Phi_T^M \uparrow i$ agree until $\bar{\tau}+1$, which implies, by def. of stable model, that M and N agree until $\bar{\tau}+1$ and hence a contradiction.

The proof is by induction on i. The base case is trivial. For the induction step, note that since T is safe, the membership of an arbitrary (t, A) with $\ell(t, A) \leq \bar{\tau} + 1$ to $\Phi_T^N \uparrow i$ depends only on conditions of the following form:

1. $(t', A') \in \Phi_T^N \uparrow i - 1$, where $\ell(t', A') \leq \bar{\tau} + 1$;
2. $(t'', A'') \notin N$, where $\ell(t'', A'') \leq \bar{\tau}$.

By induction hypothesis, $\Phi_T^N \uparrow i - 1$ and $\Phi_T^M \uparrow i - 1$ agree until $\bar{\tau} + 1$, therefore N can be equivalently replaced by M in point 1; moreover, since N and M agree until $\bar{\tau}$ by assumption, N can be equivalently replaced by M in point 2. It follows immediately that $(t, A) \in \Phi_T^N \uparrow i$ implies $(t, A) \in \Phi_T^M \uparrow i$. The opposite implication can be proved symmetrically. This completes the proof. □

Object-Oriented Databases Session

Wednesday, July 28, 1999

Chair: Martin Olivier
Rand Afrikaans University

The Security Problem against Inference Attacks on Object-Oriented Databases

Yasunori ISHIHARA† Toshiyuki MORITA‡† Minoru ITO†

† Graduate School of Information Science
Nara Institute of Science and Technology
8916-5, Takayama, Ikoma, Nara, 630-0101 Japan

‡ Systems Development Laboratory
Hitachi, Ltd.
890 Kashimada, Saiwai, Kawasaki, Kanagawa, 211-8567 Japan

Abstract

Inference attacks mean that a user infers (or tries to infer) the result of an unauthorized method execution using only authorized methods to the user. We say that a method m is secure against inference attacks by a user u if there exists no database instance for which u can infer the result of m. It is important for database administrators to know which methods are secure and which ones are not. When an administrator finds that a method which retrieves top secret information is not secure against inference attacks by u, the administrator can prevent u from attacking the method by changing the authorization for u. This paper formalizes the security problem (i.e., to determine whether a given method is secure or not) for method schemas, and presents the following results. First, it is shown that the security problem is undecidable. Next, a decidable sufficient condition for a given method to be secure is proposed. Furthermore, it is shown that the sufficient condition is also a necessary one if a given schema is monadic (i.e., every method has exactly one parameter). The time complexity to decide the condition is also evaluated. For a monadic schema, the condition is decidable (and therefore, the security problem is solvable) in polynomial time of the size of the schema.

1 Introduction

In recent years, various authorization models for object-oriented databases (OODBs) have been proposed and studied. Among of them, the method-based authorization model [5, 13] is one of the most elegant models since it is in harmony with the concept that "an object can be accessed only via its methods" in the object-oriented paradigm. In the model, an authorization A for a user u can be represented as a set of expressions $m(c_1, \ldots, c_n)$, which means that u can *directly* invoke method m on any tuple $(o_1, \ldots, o_n)$ of objects such that o_i is an object of class c_i $(1 \leq i \leq n)$. On the other hand, even if $m(c_1, \ldots, c_n) \notin A$, u can invoke m *indirectly* through another method execution in several models (e.g., protection mode in [3]). Although such indirect invocations are useful for data hiding [3], they may also allow *inference attacks* in some situations.

Example 1: Let Employee, Host, and Room be classes representing employees, hosts, and rooms, respectively. Suppose that a method uses returns the host which a given employee uses, and a method located returns the room in which a given host is placed. Also suppose that a method office, which returns the room occupied by a given employee, is implemented as office(x) = located(uses(x)).

Now suppose that the physical computer network is top secret information. In this case, an authorization for a user u may be the one shown in Fig. 1, where a solid (resp. dotted) arrow denotes an authorized (resp. unauthorized) method to u. Suppose that u have obtained that uses(John) = mars and office(John) = A626 using the authorized methods. Also suppose that u knows the implementation body of office as its behavioral specification. Then, u can infer that located(mars) = A626.

On the other hand, suppose that method uses retrieves top secret information and therefore the authorization of u is set as shown in Fig. 2. Then, u knows that located(mars) = A626, office(John) = A626, and office(x) = located(uses(x)), similarly to the former case. However, u cannot conclude that uses(John) = mars only from the above information, since there may be another host h such that uses(John) = h and located(h) = A626. □

For a given database schema S and an authorization A for a user u, an n-ary method m is said to be *secure* at $(c_1, \ldots, c_n)$ (each c_i is a class in S) against inference attacks by u if u cannot infer the result of $m(o_1, \ldots, o_n)$ for any objects o_i of class c_i in any database instance I of S, using only authorized methods to u. Otherwise, m is *insecure*. For example, if uses(Employee) and

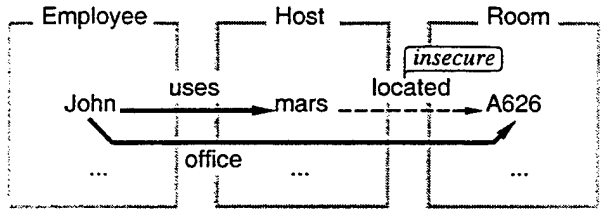


Fig. 1: An example of an insecure method.

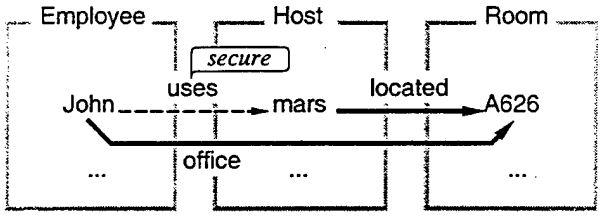


Fig. 2: An example of a secure method.

office(Employee) are authorized, then located is insecure since the user can infer located(mars) = A626 under the database instance shown in Fig. 1. On the other hand, it will be shown later that uses is secure when only located(Host) and office(Employee) are authorized. It is important for database administrators to know which methods are secure and which ones are not. When an administrator finds that a method which retrieves top secret information is insecure against inference attacks by u, the administrator can prevent u from attacking the method by changing the authorization for u.

In this paper, we formally define the security problem, i.e., to determine whether a given method is secure or not. We adopt method schemas proposed by [2] as a formal model of OODB schemas since they support such basic features of OODBs as method overloading, dynamic binding, and complex objects. The semantics is simply defined based on term rewriting. Under this formalization, we first show that the problem is undecidable. Next, we propose a decidable sufficient condition for a method to be secure. Furthermore, we show that the sufficient condition is also a necessary one if a given schema is monadic (i.e., every method has exactly one parameter). Finally, we evaluate the time complexity of deciding the condition. For a monadic method schema, the proposed condition is decidable (and therefore, the security problem is solvable) in polynomial time of the size of the schema.

The main idea of the proposed sufficient condition is to "conservatively" approximate the user's inference. The user's inference is object-level, while the approximation is class-level inference. To do the class-level inference, we use a static type checking technique for method schemas. Let m be an n-ary method and $c_1, \ldots, c_n$ be classes. The most difficult task in type checking is to solve the following question:

> Suppose that m is executed on arbitrary objects $o_1, \ldots, o_n$ such that o_i belongs to class c_i. What class does the object obtained by the execution belong to?

Unfortunately, this question is unsolvable in general [2]. However, the type checking algorithm proposed in [12] can compute a set of classes which contain all the correct classes, although the set may contain some wrong classes. Using this algorithm, we can conservatively approximate user's inference.

In this paper we discuss *precise inference* in OODBs. Precise inference means that a user can infer (or, is interested in) only the exact value of the result of an unauthorized method. On the other hand, most of the recent researches concentrate on *imprecise inference* in relational databases, not OODBs. Imprecise inference means that a user can infer (or, is interested in) possible values of the result of an unauthorized method (query) with a certain probability. In [6], FD-based imprecise inference involving abduction and partial deduction is discussed. In [15], a quantitative measure of inference risk is formally defined. Imprecise inference with external, common sense knowledge can be regarded as data mining [7, 11].

[14] focuses on both precise and imprecise inference in OODBs. Besides inferability of the result of a method execution, the article introduces the notion of controllability, which means that a user can control (alter arbitrarily) an attribute-value of an object in a database instance. We do not consider controllability since our query language does not support update operations for database instances. However, since our query language supports recursion while the one in [14] does not, detecting inferability in our formalization is not trivial. [14] also proposes, for a given database schema S and an authorization A, a sound algorithm for detecting inferability or controllability. However, [14] does not evaluate the complexity of the algorithm.

This paper is organized as follows. In Section 2, we give the definition of method schemas. In Section 3, we discuss inference attacks and formulate the security problem. We also show that the problem is undecidable. In Section 4, we propose a sufficient condition for a method to be secure. Finally, in Section 5, we conclude this paper.

2 Method Schemas

2.1 Syntax

We introduce some notations before defining the syntax of method schemas. Let F be a family of disjoint sets F_0, F_1, F_2,..., where F_n ($n = 0, 1, 2, \ldots$) is a set of function symbols of arity n. For a countable set X of variables, let $T_F(X)$ denote the set of all the terms freely generated by F and X. For a set V, let V^n denote the Cartesian product $\underbrace{V \times \cdots \times V}_{n}$. For a term $t \in T_F(X)$, an n-tuple $\mathbf{t} = (t_1, \ldots, t_n) \in (T_F(X))^n$ of terms, and an n-tuple $\mathbf{x} = (x_1, \ldots, x_n) \in X^n$ of variables, let $t[\mathbf{t}/\mathbf{x}]$ denote the term obtained by simultaneously replacing every x_i in t with t_i ($1 \leq i \leq n$). For example, $f(x_1, g(x_1, x_2))[(f(a), x_1)/(x_1, x_2)] = f(f(a), g(f(a), x_1))$. For a term t, define the set of *occurrences* $R(t)$ as the smallest set of sequences of positive integers with the following two properties:

- $\varepsilon \in R(t)$, where ε is the empty sequence.
- If $r \in R(t_i)$, then $i \cdot r \in R(f(t_1, \ldots, t_n))$ ($1 \leq i \leq n$), where the center dot "$\cdot$" represents the concatenation of sequences.

An occurrence of t specifies (the position of) a subterm of t. For example, $1 \cdot 2$ of $f(f(x, g(x)), g(x))$ specifies the first $g(x)$. The replacement in t of t' at r, denoted $t[r \leftarrow t']$, is defined as follows:

- $t[\varepsilon \leftarrow t'] = t'$;
- $f(t_1, \ldots, t_i, \ldots, t_n)[i \cdot r \leftarrow t']$
  $= f(t_1, \ldots, t_{i-1}, t_i[r \leftarrow t'], t_{i+1}, \ldots, t_n)$,
  where $1 \leq i \leq n$.

For example,

$$f(f(x, g(x)), g(x))[1 \cdot 2 \leftarrow h(a, b)] = f(f(x, h(a, b)), g(x)).$$

We go on to the definition of method schemas. Let C be a finite set of *class names* (or simply classes) and M a family of mutually disjoint finite sets M_0, M_1, M_2,..., where M_n ($n = 0, 1, 2, \ldots$) is a set of *method names* of arity n. Each M_n is partitioned into $M_{\mathrm{b},n}$ and $M_{\mathrm{c},n}$: Each $m_\mathrm{b} \in M_\mathrm{b}$ ($= \bigcup_{n \geq 0} M_{\mathrm{b},n}$) (resp. $m_\mathrm{c} \in M_\mathrm{c}$ ($= \bigcup_{n \geq 0} M_{\mathrm{c},n}$)) is called a *base method name* (resp. *composite method name*). Furthermore, each $m \in M$ ($= M_\mathrm{b} \cup M_\mathrm{c}$) is simply called a *method name*. We say that M is a *method signature*.

Hereafter, we often use a bold letter $\mathbf{v}$ to mean $(v_1, \ldots, v_n)$ without explicitly defining it when n is irrelevant or obvious from the context.

Definition 1: Let $\mathbf{c} \in C^n$. A *base method definition* of $m_\mathrm{b} \in M_{\mathrm{b},n}$ at $\mathbf{c}$ is a pair $(m_\mathrm{b}(\mathbf{c}), c)$, where $c \in C$. A *composite method definition* of $m_\mathrm{c} \in M_{\mathrm{c},n}$ at $\mathbf{c}$ is a pair $(m_\mathrm{c}(\mathbf{c}), t)$, where $t \in T_M(\{x_1, \ldots, x_n\})$. □

Let o_i be an object of class c_i ($1 \leq i \leq n$) (see Defs. 2 and 4 for formal definitions). Informally, the above base method definition declares that the application of m_b to $\mathbf{o} = (o_1, \ldots, o_n)$ results in an object of c or its subclass, while the above composite method definition states that the application of m_c to $\mathbf{o}$ results in term rewriting starting from $t[\mathbf{o}/\mathbf{x}]$. The formal definition is presented in Section 2.2.

Definition 2: A *method schema* [1, 2] S is a 5-tuple $(C, \leq, M, \Sigma_\mathrm{b}, \Sigma_\mathrm{c})$, where:

1. C is a finite set of class names.
2. $\leq$ is a partial order representing a class hierarchy. When $c' \leq c$, we say that c' is a subclass of c and c is a superclass of c'. We naturally extend $\leq$ to n-tuples of classes as follows: For two tuples $\mathbf{c} = (c_1, \ldots, c_n)$ and $\mathbf{c}' = (c'_1, \ldots, c'_n)$, we write $\mathbf{c} \leq \mathbf{c}'$ *iff* $c_i \leq c'_i$ for all i.
3. M is a method signature.
4. Σ_b is a set of base method definitions.
5. Σ_c is a set of composite method definitions.

For each possible combination $\mathbf{c} \in C^n$ and $m \in M_n$, there must exist at most one method definition of m at $\mathbf{c}$. If every method of S has exactly one parameter, then S is *monadic*. □

Example 2: An example of a method schema S_1 is shown in Fig. 3. Manager is a subclass of Employee, and Server is a subclass of Host. Method boss(e) returns the direct boss of employee e, and method supervisor(e) returns the "second least manager" among the indirect bosses of e. Since every method has arity one, S_1 is monadic. □

2.2 Semantics

Method definitions are inherited along the class hierarchy.

Definition 3: Let $S = (C, \leq, M, \Sigma_\mathrm{b}, \Sigma_\mathrm{c})$, $m_\mathrm{b} \in M_{\mathrm{b},n}$, and $\mathbf{c} \in C^n$. Suppose that $(m_\mathrm{b}(\mathbf{c}'), c') \in \Sigma_\mathrm{b}$ is the base method definition of m_b at the smallest $\mathbf{c}'$ above $\mathbf{c}$, i.e., whenever $(m_\mathrm{b}(\mathbf{c}''), c'') \in \Sigma_\mathrm{b}$ and $\mathbf{c} \leq \mathbf{c}''$, it is the case that $\mathbf{c}' \leq \mathbf{c}''$. We define the *resolution* $Res(m_\mathrm{b}(\mathbf{c}))$ of m_b at $\mathbf{c}$ as c'. If such a unique base method definition does not exist, then $Res(m_\mathrm{b}(\mathbf{c}))$ is *undefined*, denoted $\perp$. The resolution of a composite method is defined in the same way. □

C = {Employee, Manager, Host, Server, Room}

Manager $\leq$ Employee, Server $\leq$ Host

M = {boss, uses, located, supervisor, office}

Σ_b = {(boss(Employee), Employee),
(boss(Manager), Manager),
(uses(Employee), Host),
(located(Host), Room)}

Σ_c = {(supervisor(Employee), supervisor(boss(x_1))),
(supervisor(Manager), boss(x_1)),
(office(Employee), located(uses(x_1)))}

Fig. 3: A method schema S_1.

Example 3: Consider schema S_1 shown in Fig. 3. By Def. 3, *Res*(located(Server)) = Room. In other words, class Server inherits method definition (located(Host), Room) $\in \Sigma_b$. On the other hand, *Res*(boss(Server)) = $\perp$ since no superclass of Server has a definition of boss. □

The semantics of a method schema is defined as follows. To each class name, a set of objects is assigned. Also, to each base method name m_b, a mapping over appropriate sets of objects is assigned as its interpretation. The semantics of a composite method is defined by the interpretation of base methods and term rewriting.

Definition 4: An *interpretation* (or, also called a *database instance*) of a method schema S is a pair $I = (\nu, \mu)$ with the following properties:

1. To each $c \in C$, ν assigns a finite disjoint set $\nu(c)$ of *object identifiers* (or simply, *objects*). Each $o \in \nu(c)$ is called an object of class c. Let $O_I = \bigcup_{c \in C} \nu(c)$. For $\mathbf{c} = (c_1, \ldots, c_n)$, let $\nu(\mathbf{c})$ denote $\nu(c_1) \times \cdots \times \nu(c_n)$.
2. For each $m_b \in M_{b,n}$, $\mu(m_b)$ is a partial mapping from O_I^n to O_I which satisfies the following two conditions. Let $\mathbf{c}, \mathbf{c}' \in C^n$.
   (a) If $Res(m_b(\mathbf{c})) = c'$, then $\mu(m_b) \mid_{\nu(\mathbf{c})}$ is a total mapping to $\bigcup_{c \leq c'} \nu(c)$, where "|" denotes that the domain of μ is restricted to $\nu(\mathbf{c})$.
   (b) If $Res(m_b(\mathbf{c})) = \perp$, then $\mu(m_b)$ is undefined everywhere in $\nu(\mathbf{c})$. □

A term in $T_M(O_I)$ is called an *instantiated term*. That is, an instantiated term consists of method names in M and objects in O_I. The *one-step execution relation* $\rightarrow_I$ on the instantiated terms, based on the leftmost innermost reduction strategy, is defined as follows:

Definition 5: Let $m(\mathbf{o})$ ($\mathbf{o} \in \nu(\mathbf{c})$) be the subterm of $t \in T_M(O_I)$ at the leftmost innermost occurrence r.

1. If $m \in M_b$ and $Res(m(\mathbf{c})) \neq \perp$, then
   $t \rightarrow_I t[r \leftarrow \mu(m)(\mathbf{o})]$.
2. If $m \in M_c$ and $Res(m(\mathbf{c})) = t' \neq \perp$, then
   $t \rightarrow_I t[r \leftarrow t'[\mathbf{o}/\mathbf{x}]]$. □

Note that, by Def. 5, for any instantiated term t, there exists at most one term t' such that $t \rightarrow_I t'$. That is, every execution is deterministic.

Let $\rightarrow_I^*$ be the reflexive and transitive closure of $\rightarrow_I$. If $t \rightarrow_I^* t'$ and there exists no t'' such that $t' \rightarrow_I t''$, then t' is called the *execution result* of t, and we write $t\downarrow = t'$. If $t\downarrow \in O_I$, then the execution of t is *successful*, and if $t\downarrow \notin O_I$ because of nonexistence of the resolution, then the execution of t is *aborted*. In both cases (i.e., if $t\downarrow$ exists), the execution of t is *terminating*. On the other hand, if $t\downarrow$ does not exist, then the execution of t is *nonterminating*. We omit the subscript I and simply write $\rightarrow$ or $\rightarrow^*$ if I is understood from the context.

Example 4: An example of an interpretation $I_1 = (\nu_1, \mu_1)$ of S_1 is shown in Fig. 4. ν_1 is represented by gray rectangles, e.g., ν_1(Employee) = {Alice, John}. μ_1 is represented by arrows, e.g., μ_1(boss)(John) = Alice, μ_1(uses)(John) = mars. By Def. 5, supervisor(Alice) is executed as follows:

supervisor(Alice) $\rightarrow_{I_1}$ supervisor(boss(Alice))
$\rightarrow_{I_1}$ supervisor(Sara)
$\rightarrow_{I_1}$ boss(Sara)
$\rightarrow_{I_1}$ Bob.

Thus supervisor(Alice)$\downarrow$ = Bob. □

3 Inference Attacks

3.1 Authorization

Various sophisticated method-based authorization models for OODBs have been proposed. In this paper, however, discussing authorization models is not our main purpose, and therefore we adopt the following simple but general authorization model.

Definition 6: Let $S = (C, \leq, M, \Sigma_b, \Sigma_c)$. An *authorization* A for a user u under S is a finite set of $m(\mathbf{c})$, where $m \in M_n$ and $\mathbf{c} \in C^n$. Intuitively, $m(\mathbf{c}) \in A$ means that u is authorized to directly invoke method m on any tuple $\mathbf{o}$ of objects such that $\mathbf{o} \in \nu(\mathbf{c})$. □

An authorization is often modeled as a pair of a base authorization and a set of inference rules. An example of an

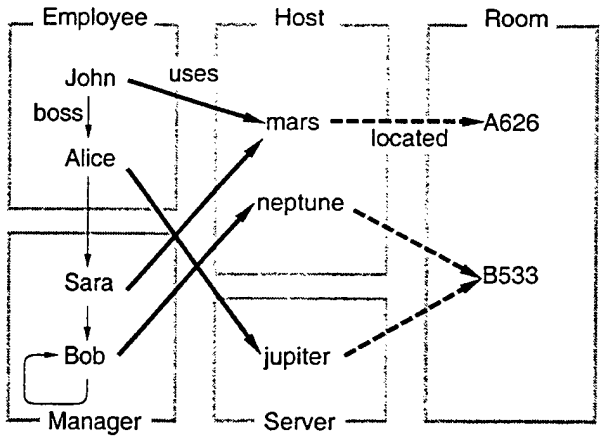


Fig. 4: An interpretation I_1 of S_1.

inference rule is "if u is authorized to invoke m on objects of c, then u is also authorized to invoke m on objects of the subclasses of c." When $c_1 \leq c$ and $c_2 \leq c$, the base authorization $\{m(c)\}$ is expanded into $\{m(c), m(c_1), m(c_2)\}$ by this rule. In this paper, we assume that a given authorization has already been expanded.

Example 5: Define an authorization A_1 for a user u under S_1 in Fig. 3 as follows:

$$A_1 = \{\text{uses(Employee)}, \text{supervisor(Employee)}, \text{supervisor(Manager)}, \text{office(Employee)}, \text{office(Manager)}\}.$$

Consider the interpretation I_1 in Fig. 4. Executing office(John) by u is permitted since John $\in \nu_1$(Employee) and office(Employee) $\in A_1$. On the other hand, executing uses(Sara) is prohibited since Sara $\in \nu_1$(Manager) but uses(Manager) $\notin A_1$. □

3.2 Formal Definition of User's Inference

In this paper, information provided by a database is modeled as a set of (in)equalities. For example, suppose that a user u executes office(John) and obtains the result A626. In this case, the information that u obtains is office(John)$\downarrow$ = A626. In what follows, we demonstrate that user's inference can be formalized as the congruence closure of a finite set of ground equalities when the two reasonable conditions (Q1) and (Q2) stated below are satisfied.

First of all, we define the information which u can obtain directly from a database instance $I = (\nu, \mu)$.

(*1) User u obtains $m(\mathbf{o})\downarrow = o$ *iff* it is the case that $m(\mathbf{c}) \in A$, $\mathbf{o} \in \nu(\mathbf{c})$, and $m(\mathbf{o})\downarrow = o \in O_I$. That is, u knows what the result of $m(\mathbf{o})$ is if executing $m(\mathbf{o})$ is authorized and the execution is successful.

(*2) User u obtains $Res(m(\mathbf{c})) = t$ *iff* it is the case that $m(\mathbf{c}) \in A$ and $Res(m(\mathbf{c})) = t$. That is, u knows the type declaration of m at $\mathbf{c}$ (when m is a base method) or the behavioral specification of m at $\mathbf{c}$ (when m is a composite method), if executing $m(\mathbf{o})$ ($\mathbf{o} \in \nu(\mathbf{c})$) is authorized.

In Example 1, (*1) and (*2) are stated informally.

Next, suppose that u can use at least four inference rules: reflexivity, symmetry, transitivity, and substitutivity (i.e., if $t_i = t_i'$ for all i, then $f(\mathbf{t}) = f(\mathbf{t}')$). Also suppose that user u knows that $o \neq o'$ for distinct objects o and o' (e.g., u knows John $\neq$ Alice, Sara $\neq$ A626, and so on).

As stated in Section 1, the goal of inference attacks is to obtain o such that $m(\mathbf{o})\downarrow = o$ for some m and $\mathbf{o}$. In other words, u wants to infer equalities. Therefore, if we find a reasonable condition under which inequalities are useless to infer equalities, we can formalize user's inference as the congruence closure of equalities induced by (*1) and (*2). Let us examine the following example:

Example 6: Recall the second case of Example 1, where u cannot infer uses(John)$\downarrow$ = mars since there may be another host h such that uses(John)$\downarrow$ = h and located(h)$\downarrow$ = A626. However, if u knows that located(o)$\downarrow \neq$ A626 for *any other object o in the database instance*, then u can conclude that uses(John) = mars. □

The above example suggests that inequalities are useless to infer equalities if the following condition is satisfied:

(Q1) User u does not know what O_I is.

In many cases, this condition is satisfied by just hiding O_I from the user.

Equalities obtained by (*2) are not ground (i.e., include variables). However, together with the following condition (Q2), they are equivalent to a finite set of ground equalities, which has many good properties:

(Q2) The user does not know what C is.

This condition is also satisfied by just hiding C.

Example 7: Consider a schema with a composite method m_c which has the same resolution t at every class $c \in C$. Let $A = \{m_c(c) \mid c \in C\}$ be an authorization for a user u.

Assume that u knows what C is. Then, u can infer that $m_c(t')\downarrow = t[t'/x]\downarrow$ for any term t' such that $t'\downarrow \in O_I$, since

m_c has the same resolution t at any class. Note that, in this inference, u does not need to know which class $t'\downarrow$ belongs to.

On the other hand, if (Q2) is satisfied, then u cannot conclude that $m_c(t')\downarrow = t[t'/x]\downarrow$ without exactly inferring the class to which $t'\downarrow$ belongs since there may be another class c in C such that $t'\downarrow \in \nu(c)$ and $Res(m_c(c)) \neq t$. □

Type checking [2, 12] is useless when (Q2) is satisfied. Therefore, to know the class to which $t'\downarrow$ belongs is to infer the exact value of $t'\downarrow$. Thus, the equalities obtained by (*2) can be applied only to terms t' such that $t'\downarrow$ is known. This means that each equality $Res(m_c(\mathbf{c})) = t$ obtained by (*2) can be regarded as $\{m_c(\mathbf{o})\downarrow = t[\mathbf{o}/\mathbf{x}]\downarrow \mid \mathbf{o} \in \nu(\mathbf{c})\}$, which is a finite set of ground equalities.

Consequently, by assuming (Q1) and (Q2), we can model user's inference as the congruence closure of a finite set of ground equalities induced by (*1) and (*2). For technical reasons, we define the congruence closure through rewriting rules $\rhd_{I,A}$ introduced below. From the correctness of Knuth-Bendix completion [10], $t\downarrow = o$ *iff* t is reducible to o by $\rhd_{I,A}$.

Definition 7: Define $P_{I,A}$ as the minimum set of rewriting rules $\rhd_{I,A}$ on $T_M(O_I)$ satisfying the following three conditions. Intuitively, $t \rhd_{I,A} o$ means that the user knows or can infer that $t\downarrow = o$.

(A) If $m(\mathbf{c}) \in A$, $\mathbf{o} \in \nu(\mathbf{c})$, and $m(\mathbf{o})\downarrow = o \in O_I$, then $P_{I,A}$ contains

$$m(\mathbf{o}) \rhd_{I,A} o.$$

This corresponds to (*1).

(B) If $m_c(\mathbf{c}) \in A$, $m_c \in M_c$, $\mathbf{o} \in \nu(\mathbf{c})$, $m_c(\mathbf{o})\downarrow = o \in O_I$, and $Res(m_c(\mathbf{c})) = t \neq \perp$, then $P_{I,A}$ contains

$$t[\mathbf{o}/\mathbf{x}] \rhd_{I,A} o.$$

This essentially corresponds to (*2).

(C) If $P_{I,A}$ contains $t \rhd_{I,A} o$ and $t'' \rhd_{I,A} o''$ such that t'' is a proper subterm of t at τ'', then $P_{I,A}$ contains

$$t[\tau'' \leftarrow o''] \rhd_{I,A} o.$$

See also Fig. 5. This simulates Knuth-Bendix completion procedure.

By definition, the right-hand side of each rule is an object. Note that the existence of $t \rhd_{I,A} o$ in $P_{I,A}$ implies $t \rightarrow_I^* o$.

Define $\Rightarrow_{I,A}$ as the one-step reduction relation by $\rhd_{I,A}$. That is, $t \Rightarrow_{I,A} t'$ *iff* there exists a subterm t'' of t at τ'' such that $t'' \rhd_{I,A} o'' \in P_{I,A}$ and $t' = t[\tau'' \leftarrow o'']$. Let

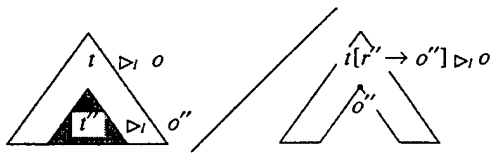


Fig. 5: Condition (C) of Definition 7.

$\Rightarrow_{I,A}^*$ denote the reflexive and transitive closure of $\Rightarrow_{I,A}$. For readability, we often write $\rhd_I$ and P_I instead of $\rhd_{I,A}$ and $P_{I,A}$, respectively. □

Example 8: For S_1 in Fig. 3, I_1 in Fig. 4, and A_1 in Example 5, P_{I_1} is computed as shown in Fig. 6. Rules (A1)–(A10) are obtained by Def. 7(A), and (B1)–(B8) by Def. 7(B) with composite methods supervisor and office. Rules (C1) and (C2) are obtained by Def. 7(C). For example, (C1) is derived from (A1) and (B5).

Rule (C1) indicates that the user can infer that located(mars)↓ = A626, as stated in the first case of Example 1. Moreover, rule (C2) indicates that located even for a server jupiter can be inferred. Let τ = office(boss(Sara)) and τ' = office(boss(John)). Then,

$$\tau \Rightarrow_{I_1} \text{office(Bob)} \Rightarrow_{I_1} \text{B533}.$$

Thus user u can infer that $\tau\downarrow$ = B533. On the other hand, u cannot infer the value of $\tau'\downarrow$ (although $\tau'\downarrow$ = B533) since no subterm of τ' can be rewritten by the rules in P_{I_1}. □

3.3 The Security Problem

The notion of security of methods discussed in Section 1 is naturally extended to terms in $T_M(C)$ as follows: A term $\tau \in T_M(C)$ is said to be *secure* if there exists no interpretation $I = (\nu, \mu)$ such that $\tau[\mathbf{o}/\mathbf{c}] \Rightarrow_{I,A}^* o$ for any $\mathbf{o} \in \nu(\mathbf{c})$ and $o \in O_I$. Otherwise, τ is *insecure*. The *security problem* is to determine whether a given $\tau \in T_M(C)$ is secure or not.

Theorem 1: The security problem for method schemas with methods of arity two is undecidable.

Sketch of Proof: The *type-consistency problem* is to determine whether, for a given method schema S, there exists an interpretation of S which causes an aborted execution. [8] shows that the problem for method schemas with methods of arity two is undecidable by reducing the Post's Correspondence Problem (PCP) to the problem. In the reduction, each interpretation I is regarded as a candidate for a solution to a PCP. If I is actually a solution, then execution

uses(John)	$\rhd_{I_1}$	mars	(A1)
uses(Alice)	$\rhd_{I_1}$	jupiter	(A2)
supervisor(John)	$\rhd_{I_1}$	Bob	(A3)
supervisor(Alice)	$\rhd_{I_1}$	Bob	(A4)
supervisor(Sara)	$\rhd_{I_1}$	Bob	(A5)
supervisor(Bob)	$\rhd_{I_1}$	Bob	(A6)
office(John)	$\rhd_{I_1}$	A626	(A7)
office(Alice)	$\rhd_{I_1}$	B533	(A8)
office(Sara)	$\rhd_{I_1}$	A626	(A9)
office(Bob)	$\rhd_{I_1}$	B533	(A10)
supervisor(boss(John))	$\rhd_{I_1}$	Bob	(B1)
supervisor(boss(Alice))	$\rhd_{I_1}$	Bob	(B2)
boss(Sara)	$\rhd_{I_1}$	Bob	(B3)
boss(Bob)	$\rhd_{I_1}$	Bob	(B4)
located(uses(John))	$\rhd_{I_1}$	A626	(B5)
located(uses(Alice))	$\rhd_{I_1}$	B533	(B6)
located(uses(Sara))	$\rhd_{I_1}$	A626	(B7)
located(uses(Bob))	$\rhd_{I_1}$	B533	(B8)
located(mars)	$\rhd_{I_1}$	A626	(C1)
located(jupiter)	$\rhd_{I_1}$	B533	(C2)

Fig. 6: Contents of P_{I_1}.

of a term, say $m(o)$, is aborted under I. Otherwise, $m(o)$ is nonterminating. By slightly modifying the reduction in [8], we can construct a schema with the following properties:

- If I is a solution, then the execution of a term, say $m'(o)$, is successful under I.
- Otherwise, $m'(o)$ is nonterminating under I.

Let c be the class to which o belongs. Let $A = \{m'(c)\}$ and $\tau = m'(c)$. Then, the PCP has a solution *iff* there exists $I = (\nu, \mu)$ such that $\tau[o/c] \Rightarrow_I^* o'$ for some o and o'. □

4 Security Analysis

4.1 A Sufficient Condition

In this section we propose a decidable sufficient condition for a given term $\tau \in T_M(C)$ to be secure. The main idea is to introduce new rewriting rules on $T_M(C)$ which "conservatively" approximate $\rhd_{I,A}$, i.e., if τ is insecure, then τ is reducible to a class c by the new rewriting rules. Intuitively, each $t \in T_M(C)$ is considered as the set of instantiated terms $\{t[\mathbf{o}/\mathbf{c}] \mid \mathbf{o} \in \nu(\mathbf{c})\}$. The "execution result" $E(t)$ of t is defined as follows: $c \in E(t)$ *iff* there exists an interpretation $I = (\nu, \mu)$ such that $t[\mathbf{o}/\mathbf{c}]\downarrow \in \nu(c)$ for some $\mathbf{o} \in \nu(\mathbf{c})$. Unfortunately, we cannot compute E exactly in general [2]. However, we can compute $Z : T_M(C) \to 2^C$ such that $Z(t) \supseteq E(t)$ for every $t \in T_M(C)$ [12]. We use such Z to approximate $\rhd_{I,A}$. The smaller $Z(t)$ is, the better approximation we have, although the approximation is still conservative even when $Z(t) = C$ for all t. The algorithm in [12] gives a fairly small Z.

$$
\begin{aligned}
Z(\text{boss(Employee)}) &= \{\text{Employee, Manager}\}\\
Z(\text{boss(Manager)}) &= \{\text{Manager}\}\\
Z(\text{uses(Employee)}) &= \{\text{Host, Server}\}\\
Z(\text{uses(Manager)}) &= \{\text{Host, Server}\}\\
Z(\text{located(Host)}) &= \{\text{Room}\}\\
Z(\text{located(Server)}) &= \{\text{Room}\}\\
Z(\text{supervisor(Employee)}) &= \{\text{Manager}\}\\
Z(\text{supervisor(Manager)}) &= \{\text{Manager}\}\\
Z(\text{office(Employee)}) &= \{\text{Room}\}\\
Z(\text{office(Manager)}) &= \{\text{Room}\}\\
Z(m(c)) &= \emptyset \text{ for any other combinations of } m \text{ and } c,\\
Z(m(t)) &= \bigcup_{c \in Z(t)} Z(m(c))
\end{aligned}
$$

Fig. 7: Z for schema S_1.

Example 9: Using the algorithm in [12], we can compute Z for schema S_1 in Fig. 3. The result is presented in Fig. 7. For example, Z(boss(Employee)) = {Employee, Manager} means that for any object e of Employee, the result of boss(e) is an object of either Employee or Manager. Actually, the obtained Z is equal to E in this case. □

The next definition introduces the new rewriting rules $\rhd_{S,A,Z}$ on $T_M(C)$ which approximate $\rhd_{I,A}$.

Definition 8: Define $P_{S,A,Z}$ as the minimum set of rewriting rules $\rhd_{S,A,Z}$ on $T_M(C)$ satisfying the following three conditions:

(A) If $m(\mathbf{c}) \in A$, then $P_{S,A,Z}$ contains

$$m(\mathbf{c}) \rhd_{S,A,Z} c$$

for each $c \in Z(m(\mathbf{c}))$.

(B) If $m_c(\mathbf{c}) \in A$, $m_c \in M_{c,n}$, and $Res(m_c(\mathbf{c})) = t \neq \perp$, then $P_{S,A,Z}$ contains

$$t[\mathbf{c}/\mathbf{x}] \rhd_{S,A,Z} c$$

uses(Employee)	$\rhd_{S_1}$	Host	(Ai)
uses(Employee)	$\rhd_{S_1}$	Server	(Aii)
supervisor(Employee)	$\rhd_{S_1}$	Manager	(Aiii)
supervisor(Manager)	$\rhd_{S_1}$	Manager	(Aiv)
office(Employee)	$\rhd_{S_1}$	Room	(Av)
office(Manager)	$\rhd_{S_1}$	Room	(Avi)
supervisor(boss(Employee))	$\rhd_{S_1}$	Manager	(Bi)
boss(Manager)	$\rhd_{S_1}$	Manager	(Bii)
located(uses(Employee))	$\rhd_{S_1}$	Room	(Biii)
located(uses(Manager))	$\rhd_{S_1}$	Room	(Biv)
located(Host)	$\rhd_{S_1}$	Room	(Ci)
located(Server)	$\rhd_{S_1}$	Room	(Cii)

Fig. 8: Contents of P_{S_1}.

for each $c \in Z(t[\mathbf{c}/\mathbf{x}])$.

(C) If P_S contains $t \rhd_{S,A,Z} c$ and $t'' \rhd_{S,A,Z} c''$ such that t'' is a proper subterm of t at r'', then $P_{S,A,Z}$ contains

$$t[r'' \leftarrow c''] \rhd_{S,A,Z} c'$$

for each $c' \in Z(t[r'' \leftarrow c''])$.

Define $\Rightarrow_{S,A,Z}$ as the one-step reduction relation by $\rhd_{S,A,Z}$. Let $\Rightarrow^*_{S,A,Z}$ denote the reflexive and transitive closure of $\Rightarrow_{S,A,Z}$. For readability, we often write $\rhd_S$ and P_S instead of $\rhd_{S,A,Z}$ and $P_{S,A,Z}$, respectively. □

Example 10: Fig. 8 presents the contents of P_{S_1} for schema S_1 in Fig. 3, A_1 in Example 5, and Z in Fig. 7. Rules (Ai)–(Avi) are obtained by Def. 8(A), and (Bi)–(Biv) by Def. 8(B) with composite methods supervisor and office. Rules (Ci) and (Cii) are obtained by Def. 8(C).

Rule (Cii) indicates that the user may be able to infer the location of a server. Moreover, rules (Avi) and (Bii) together indicate that the user may be able to infer the office of the boss of a manager. Compare this with the explanation in Example 8. □

The next lemma states that each rule in P_I is conservatively approximated by a rule in P_S.

Lemma 1: If there is an interpretation $I = (\nu, \mu)$ such that $t[\mathbf{o}/\mathbf{x}] \rhd_I o \in P_I$ for some $\mathbf{o} \in \nu(\mathbf{c})$ and $o \in \nu(c)$, then $t[\mathbf{c}/\mathbf{x}] \rhd_S c \in P_S$.

Proof: We use induction on the number of the iterations of a procedure which computes the least fixed point satisfying the three conditions in Def. 7.

Basis: Consider the case that $m(\mathbf{o}) \rhd_I o$ $(o \in \nu(c))$ is obtained from Def. 7(A). Then, $m(\mathbf{c}) \in A$, $\mathbf{o} \in \nu(\mathbf{c})$, and $m(\mathbf{o})\downarrow = o$. Moreover, $c \in Z(m(\mathbf{c}))$ from the property of Z. From Def. 8(A), P_S contains $m(\mathbf{c}) \rhd_S c$ since $m(\mathbf{c}) \in A$ and $c \in Z(m(\mathbf{c}))$. The case that $Res(m_c(\mathbf{c}))[\mathbf{o}/\mathbf{x}] \rhd_I o$ is obtained from Def. 7(B) can be proved in the same way.

Induction: Suppose that $t''[\mathbf{o}''/\mathbf{x}'']$ $(\mathbf{o}'' \in \nu(\mathbf{c}''))$ is a proper subterm of $t[\mathbf{o}/\mathbf{x}]$ $(\mathbf{o} \in \nu(\mathbf{c}))$ at r'' and that $t[\mathbf{o}/\mathbf{x}] \rhd_I o$ $(o \in \nu(c))$ and $t''[\mathbf{o}''/\mathbf{x}''] \rhd_I o''$ $(o'' \in \nu(c''))$ have been obtained. Let $t'[\mathbf{o}'/\mathbf{x}'] = t[\mathbf{o}/\mathbf{x}][r'' \leftarrow o'']$ $(\mathbf{o}' \in \nu(\mathbf{c}'))$, and suppose that $t'[\mathbf{o}'/\mathbf{x}'] \rhd_I o$ is obtained from Def. 7(C). By the inductive hypothesis, P_S contains both $t[\mathbf{c}/\mathbf{x}] \rhd_S c$ and $t''[\mathbf{c}''/\mathbf{x}''] \rhd_S c''$. From the definition of $t'[\mathbf{o}'/\mathbf{x}']$, we obtain $t'[\mathbf{c}'/\mathbf{x}'] = t[\mathbf{c}/\mathbf{x}][r'' \leftarrow c'']$. Since $t'[\mathbf{o}'/\mathbf{x}'] \rhd_I o \in P_I$ implies $t'[\mathbf{o}'/\mathbf{x}']\downarrow = o$, it holds that $c \in Z(t'[\mathbf{c}'/\mathbf{x}'])$. From the above inductive hypothesis and Def. 8(C), we can conclude that $t'[\mathbf{c}'/\mathbf{x}'] \rhd_S c \in P_S$. □

We have the following theorem:

Theorem 2: Let $\tau \in T_M(C)$. If there exists no class c such that $\tau \Rightarrow^*_{S,A,Z} c$, then τ is secure, i.e., there exists no interpretation $I = (\nu, \mu)$ such that $\tau[\mathbf{o}/\mathbf{c}] \Rightarrow^*_{I,A} o$ for any $\mathbf{o} \in \nu(\mathbf{c})$ and $o \in O_I$.

Proof: By Lemma 1, it can be easily shown that if there is $I = (\nu, \mu)$ such that $t[\mathbf{o}/\mathbf{x}] \Rightarrow^*_I t'[\mathbf{o}'/\mathbf{x}']$ for some $\mathbf{o} \in \nu(\mathbf{c})$ and $\mathbf{o}' \in \nu(\mathbf{c}')$, then $t[\mathbf{c}/\mathbf{x}] \Rightarrow^*_S t'[\mathbf{c}'/\mathbf{x}']$. The theorem is implied by this fact. □

The proposed sufficient condition is obviously decidable, since the right-hand side of each rule $\rhd_{S,A,Z}$ is a class and therefore the "size" of the term decreases every time a rule is applied.

Example 11: Consider schema S_1 in Fig. 3, and let τ = office(boss(Employee)). We can conclude that τ is secure since no subterm of τ can be rewritten by any rule P_{S_1} in Fig. 8. □

Example 12: We said that method uses is secure in the second case of Example 1. Actually, it is not difficult to see that P_S has only located(Host) $\rhd_S$ Room and office(Employee) $\rhd_S$ Room. This implies that uses(Employee) is secure. □

It is open whether the undecidability of the security problem stems only from the uncomputability of E. In other words, it is open whether or not the sufficient condition in Theorem 2 is also a necessary one when we use E as Z.

4.2 Monadic Case

When a given schema is monadic, the algorithm in [12] computes E in time polynomial of the size of S. Moreover, when a given schema is monadic and we use E as Z, the sufficient condition in Theorem 2 is also a necessary one.

Theorem 3: Let S be a monadic schema and $\tau \in T_M(C)$. If there exists a class c' such that $\tau \Rightarrow^*_{S,A,E} c'$, then τ is insecure, i.e., there exists an interpretation I such that $\tau[o/c] \Rightarrow^*_{I,A} o'$ for some $o \in \nu(c)$ and $o' \in O_I$.
Proof: See Appendix B. □

Example 13: Consider schema S_1 in Fig. 3. Since S_1 is monadic and Z presented in Fig. 7 is equal to E, we can conclude that located(Server) is insecure by rule (Cii) in Fig. 8. Moreover,

$$\begin{aligned} \text{office(boss(Manager))} &\Rightarrow_{S_1} \text{office(Manager)} \\ &\Rightarrow_{S_1} \text{Room}, \end{aligned}$$

and therefore office(boss(Manager)) is insecure. □

4.3 Complexity

We summarize the time complexity of deciding the sufficient condition stated in Theorem 2. Define the size of a term t as $|R(t)|$, i.e., the number of occurrences of t. Define the description length of Σ_c, denoted $\|\Sigma_c\|$, as the sum of the size of all t such that $(m(\mathbf{c}), t) \in \Sigma_c$. Also, define the size of S, denoted $\|S\|$, as follows:

$$\|S\| = |C| + |\leq| + |M| + |\Sigma_b| + \|\Sigma_c\|.$$

Let k be the maximum arity of all the methods. The height of t is defined as the maximum length of the occurrences in $R(t)$. Let L and H be the maximum size and height of all t in $\{t \mid (m(\mathbf{c}), t) \in \Sigma_c\} \cup \{\tau\}$, respectively.

By assuming $L \leq \|S\|$, the total time complexity (including computation of Z) is

$$O(k^{H+1} L \|S\|^2 (|C|+1)^{2k^{H+1}+1} \log \|S\|).$$

See Appendix C for details.

Theorem 4: For a monadic method schema, the security problem is solvable in polynomial time of the size of the schema. □

5 Conclusions

We have formalized the security problem against inference attacks on OODBs, and shown that the problem is undecidable. Then we have proposed a decidable sufficient condition for a given method to be secure, by introducing class-level inference ($\rhd_S$) which conservatively approximates object-level inference ($\rhd_I$). We believe that the approximation is fairly tight in spite of its simple definition, since the sufficient condition becomes a necessary one when the given schema is monadic.

There are several variants of the security problem. For example, [9] discusses the instance-level security. In [9], a method m is *secure* if a user cannot infer the result of m under a *given database instance*. The instance-level security problem is solvable in polynomial time in practical cases.

In several situations, imprecise inference becomes powerful enough to cause serious problems. Moreover, method schemas do not seem a perfect model of OODB schemas since they do not support multi-valued methods, update operations, and so on. Therefore, we intend to extend both inference and database models. Furthermore, the security of incomplete OODB schemas should be considered, so that the security can be checked in parallel with designing OODB schemas.

Acknowledgments

The authors are grateful to Professor Hiroyuki Seki of Nara Institute of Science and Technology for his invaluable discussions and comments. This research is supported in part by Japan Society for the Promotion of Science under Grant-in-Aid for Encouragement of Young Scientists No. 11780306.

References

[1] S. Abiteboul, R. Hull and V. Vianu, *Foundations of databases*, pp. 563–571, Addison-Wesley, 1995.

[2] S. Abiteboul, P. Kanellakis, S. Ramaswamy and E. Waller, "Method schemas," *JCSS*, Vol. 51, No. 3, pp. 433–455, 1995.

[3] E. Bertino and P. Samarati, "Research issues in discretionary authorizations for object bases," *Proc. OOPSLA-93 Conf. Workshop on Security for Object-Oriented Systems*, pp. 183–199, 1994.

[4] T. H. Cormen, C. E. Leiserson and R. L. Rivest, *Introduction to algorithms*, The MIT Press, 1990.

[5] E. B. Fernandez, M. M. Larrondo-Petrie and E. Gudes, "A method-based authorization model for object-oriented databases," *Proc. OOPSLA-93 Conf. Workshop on Security for Object-Oriented Systems*, pp. 135–150, 1993.

[6] J. Hale, J. Threet and S. Shenoi, "A practical formalism for imprecise inference control," *Database Security VIII: Proc. 8th DBSec*, pp. 139–156, 1994.

[7] T. H. Hinke, H. S. Delugach and R. Wolf, "A framework for inference-directed data mining," *Database Security X: Proc. 10th DBSec*, pp. 229–239, 1996.

[8] Y. Ishihara, S. Shimizu, H. Seki and M. Ito, "The type-consistency problem for queries in object-oriented databases," *NAIST Technical Report 98004*, http://isw3.aist-nara.ac.jp/IS/TechReport2/report/98004.ps, 1998.

[9] T. Morita, Y. Ishihara, H. Seki and M. Ito, "A Formal Approach to Detecting Security Flaws in Object-Oriented Databases," *IEICE Transactions on Information and Systems*, Vol. E82-D, No. 1, pp. 89–98, 1999.

[10] D. A. Plaisted, "Equational reasoning and term rewriting systems," in *Handbook of Logic in Artificial Intelligence and Logic Programming*, Vol. 1, pp. 273–364, Oxford Science Publications, 1993.

[11] S. Rath, D. Jones, J. Hale and S. Shenoi, "A tool for inference detection and knowledge discovery in databases," *Database Security IX: Proc. 9th DBSec*, pp. 229–239, 1995.

[12] H. Seki, Y. Ishihara and H. Dodo, "Testing type consistency of method schemas," *IEICE Transactions on Information and Systems*, Vol. E81-D, No. 3, 1998.

[13] H. Seki, Y. Ishihara and M. Ito, "Authorization analysis of queries in object-oriented databases," *Proc. 4th DOOD, LNCS 1013*, pp. 521–538, 1995.

[14] K. Tajima, "Static detection of security flaws in object-oriented databases," *Proc. 1996 SIGMOD*, pp. 341–352, 1996.

[15] K. Zhang, "IRI: A quantitative approach to inference analysis in relational databases," *Database Security XI: Proc. 11th DBSec*, pp. 279–290, 1997.

APPENDIX

A Notation Table

$T_M(X)$: the set of all the terms freely generated by method signature M and variables X.

$t[\mathbf{t}/\mathbf{x}]$: the term obtained by simultaneously replacing every variable $x_i \in \mathbf{x}$ in term t with $t_i \in \mathbf{t}$.

$R(t)$: the set of occurrences of term t.

$t[r \leftarrow t']$: the replacement in term t of t' at occurrence r.

$Res(m(\mathbf{c}))$: the resolution of method m at tuple $\mathbf{c}$ of classes.

O_I: the set of all the objects in interpretation I.

$\rightarrow_I$: the one-step execution relation.

$\rightarrow_I^*$: the reflexive and transitive closure of $\rightarrow_I$.

$t\downarrow$: the execution result of $t \in T_M(O_I)$.

$\rhd_{I,A}$: the rewriting rules on $T_M(O_I)$ representing user's inference on interpretation I under authorization A.

$P_{I,A}$: the set of rewriting rules $\rhd_{I,A}$.

$\Rightarrow_{I,A}$: the one-step reduction relation by $\rhd_{I,A}$.

$\Rightarrow_{I,A}^*$: the reflexive and transitive closure of $\Rightarrow_{I,A}$.

$E(t)$: the "execution result" of $t \in T_M(C)$.

$\rhd_{S,A,Z}$: the rewriting rules on $T_M(C)$ representing user's inference on an interpretation of S under authorization A, using Z which approximates E.

$P_{S,A,Z}$: the set of rewriting rules $\rhd_{S,A,Z}$.

$\Rightarrow_{S,A,Z}$: the one-step reduction relation by $\rhd_{S,A,Z}$.

$\Rightarrow_{S,A,Z}^*$: the reflexive and transitive closure of $\Rightarrow_{S,A,Z}$.

B Complete Proof of Theorem 3

We introduce a *syntactic interpretation* $I_S = (\nu_S, \mu_S)$ of a monadic schema S, and show that I_S satisfies Theorem 3. Let N be a positive integer.

1. For each $c \in C$, $\nu_S(c) = \{c \cdot \alpha \mid \alpha \in C^*$ and the length of $c \cdot \alpha$ is at most $N\}$, where C^* denotes the Kleene closure of C.
2. For each $m_b \in M_b$, define $\mu_S(m_b)$ as follows:
   (a) If $Res(m_b(c_0)) = c'$, then $\mu_S(m_b)(c_0) = c'$, and for $j \geq 1$,

$$\mu_S(m_b)(c_0 \cdot c_1 \cdot c_2 \cdots c_j) = \begin{cases} c_1 \cdot c_2 \cdots c_j & \text{if } c_1 \leq c', \\ c' \cdot c_2 \cdots c_j & \text{otherwise.} \end{cases}$$

   (b) If $Res(m_b(c_0)) = \perp$, then $\mu_S(m_b)(c_0 \cdot c_1 \cdot c_2 \cdots c_j)$ $(j \geq 0)$ is undefined.

We consider a syntactic interpretation $I_S = (\nu_S, \mu_S)$ with sufficiently large N.

In order to prove that I_S satisfies the theorem, we need several technical lemmas.

Lemma 2: Let $t \in T_M(\{x\})$, and suppose that $c' \in E(t[c/x])$. There exists $\beta \in C^*$ such that

1. the first symbol of $\beta \cdot c'$ is c, and
2. for each $\alpha \in C^*$ such that $\beta \cdot c' \cdot \alpha$ is an object of I_S (i.e., the length of $\beta \cdot c' \cdot \alpha$ is at most N),

$$t[\beta \cdot c' \cdot \alpha/x] \rightarrow_{I_S}^* c' \cdot \alpha.$$

We call β a *reduction string* of $(t[c/x], c')$.

Proof: Suppose that $c' \in E(t[c/x])$. By the definition of E, there exists an interpretation $I = (\nu, \mu)$ such that $t[o/x] \to_I^* o'$ for some $o \in \nu(c)$ and $o' \in \nu(c')$. Consider the i-th step (counting from zero) $t_i[o_i/x] \to_I t_{i+1}[o_{i+1}/x]$ of the reduction $t[o/x] \to_I^* o'$, where $t_0[o_0/x] = t[o/x]$ and $t_n[o_n/x] = o'$. Let c_i $(0 \leq i \leq n-1)$ be the class such that $o_i \in \nu(c_i)$, and $m_i(o_i)$ be the innermost term of $t_i[o_i/x]$. Define β_i $(0 \leq i \leq n-1)$ as follows:

$$\beta_i = \begin{cases} c_i & \text{if } m_i \in M_{\mathrm{b}}, \\ \varepsilon & \text{otherwise.} \end{cases}$$

In what follows, we show that $\beta = \beta_0 \cdots \beta_{n-1}$ satisfies the conditions of this lemma.

It is easily verified that β satisfies the first condition since

- $o_0 = o \in \nu(c)$, and
- if $m_i \in M_{\mathrm{c}}$, then $o_{i+1} = o_i$ by the definition of $\to_I$.

To see that β also satisfies the second condition, consider the execution of $t_0[\beta \cdot c' \cdot \alpha/x]$ under I_S for an arbitrary $\alpha \in C^*$. If $m_0 \in M_{\mathrm{b}}$, then $\beta_0 = c_0$, and thus $t_0[\beta \cdot c' \cdot \alpha/x] \to_{I_S} t_1[\beta_1 \cdots \beta_{n-1} \cdot c' \cdot \alpha/x]$. On the other hand, if $m_0 \in M_{\mathrm{c}}$, then $\beta_0 = \varepsilon$, and thus $t_0[\beta \cdot c' \cdot \alpha/x] \to_{I_S} t_1[\beta_1 \cdots \beta_{n-1} \cdot c' \cdot \alpha/x]$. In either case,

$$\begin{aligned} t_0[\beta \cdot c' \cdot \alpha/x] &= t_0[\beta_0 \cdot \beta_1 \cdots \beta_{n-1} \cdot c' \cdot \alpha/x] \\ &\to_{I_S} t_1[\beta_1 \cdots \beta_{n-1} \cdot c' \cdot \alpha/x]. \end{aligned}$$

By repeating this, $t[\beta \cdot c' \cdot \alpha/x] \to_{I_S}^* c' \cdot \alpha$. □

Lemma 3: Let t, t', $t'' \in T_M(\{x\})$ such that t'' is a subterm of t at r'' and $t' = t[r'' \leftarrow x]$. If both $c'' \in E(t''[c/x])$ and $c' \in E(t'[c''/x])$ hold, then $c' \in E(t[c/x])$.

Proof: By Lemma 2, there exist reduction strings β'' of $(t''[c/x], c'')$ and β' of $(t'[c''/x], c')$, i.e.,

- the first symbol of $\beta'' \cdot c''$ is c,
- $t''[\beta'' \cdot c'' \cdot \alpha''/x] \to_{I_S}^* c'' \cdot \alpha''$ for any $\alpha'' \in C^*$,
- the first symbol of $\beta' \cdot c'$ is c'', and
- $t'[\beta' \cdot c' \cdot \alpha'/x] \to_{I_S}^* c' \cdot \alpha'$ for any $\alpha' \in C^*$.

When we choose α'' so that $c'' \cdot \alpha'' = \beta' \cdot c' \cdot \alpha'$, we have

$$t[\beta'' \cdot \beta' \cdot c' \cdot \alpha'/x] \to_{I_S}^* c' \cdot \alpha'.$$

By the definition of E, c' must be in $E(t[c/x])$. □

Lemma 4: Let t, t', $t'' \in T_M(\{x\})$ such that t'' is a subterm of t at r'' and $t' = t[r'' \leftarrow x]$. Suppose that $c'' \in E(t''[c/x])$ and $c' \in E(t'[c''/x])$. Let β' be an arbitrary reduction string of $(t'[c''/x], c')$. Then, there exist reduction strings β of $(t[c/x], c')$ and β'' of $(t''[c/x], c'')$ such that $\beta = \beta'' \cdot \beta'$.

Proof: Let β'' and β' be arbitrary reduction strings of $(t''[c/x], c'')$ and $(t'[c''/x], c')$, respectively. By the proof of Lemma 3, $\beta'' \cdot \beta'$ is a reduction string of $(t[c/x], c')$. This fact implies the lemma. □

The next lemma states that if $t[c/x] \rhd_S c' \in P_S$, then $t[o/x] \rhd_{I_S} o' \in P_{I_S}$ for some $o \in \nu_S(c)$ and $o' \in \nu_S(c')$. In this sense, P_S contains no "wrong" rules.

Lemma 5: Let c, $c' \in C$, and $t \in T_M(\{x\})$. If $t[c/x] \rhd_S c' \in P_S$, then for an arbitrary reduction string β of $(t[c/x], c')$ and for any $\alpha \in C^*$,

$$t[\beta \cdot c' \cdot \alpha/x] \rhd_{I_S} c' \cdot \alpha \in P_{I_S}.$$

Proof: We use induction on the number of the iterations of a procedure which computes the least fixed point satisfying the three conditions in Def. 8.

Basis: Suppose that $m(c) \rhd_S c'$ is obtained from Def. 8(A). Let β be an arbitrary reduction string of $(m(c), c')$. Since $m(c) \in A$ and $m(\beta \cdot c' \cdot \alpha) \to_{I_S}^* c' \cdot \alpha$, we obtain $m(\beta \cdot c' \cdot \alpha) \rhd_{I_S} c' \cdot \alpha$ from Def. 7(A). The case that $Res(m_{\mathrm{c}}(c))[c/x] \rhd_S c'$ is obtained from Def. 8(B) can be proved similarly.

Induction: Suppose that there exist $c \in C$ and t, t', $t'' \in T_M(\{x\})$ such that

- t'' is a subterm of t at r'',
- $t' = t[r'' \leftarrow x]$,
- $t''[c/x] \rhd_S c'' \in P_S$,
- $t[c/x] \rhd_S c_1 \in P_S$ for some c_1, and
- $c' \in E(t'[c''/x])$.

Also suppose that $t'[c''/x] \rhd_S c'$ is obtained from Def. 8(C). Since $t''[c/x] \rhd_S c'' \in P_S$, it holds that $c'' \in E(t''[c/x])$ by Def. 8. By Lemma 3, it holds that $c' \in E(t[c/x])$. Then, by Def. 8 again, $t[c/x] \rhd_S c'$ must be in P_S. Let β' be an arbitrary reduction string of $(t'[c''/x], c')$. That is, the first symbol of $\beta' \cdot c'$ is c'' and $t'[\beta' \cdot c' \cdot \alpha'/x] \to_{I_S}^* c' \cdot \alpha'$ for any $\alpha' \in C^*$. By Lemma 4 and the inductive hypothesis for $t[c/x] \rhd_S c'$ and $t''[c/x] \rhd_S c''$, there exist β and β'' such that

- the first symbol of $\beta \cdot c'$ is c,
- $t[\beta \cdot c' \cdot \alpha/x] \rhd_{I_S} c' \cdot \alpha \in P_{I_S}$ for any $\alpha \in C^*$,
- the first symbol of $\beta'' \cdot c''$ is c,

- $t''[\beta''\cdot c''\cdot\alpha''/x] \rhd_{I_S} c''\cdot\alpha'' \in P_{I_S}$ for any $\alpha'' \in C^*$, and
- $\beta = \beta''\cdot\beta'$.

When we choose α and α'' so that $\beta'\cdot c'\cdot\alpha = c''\cdot\alpha''$, it holds that $\beta\cdot c'\cdot\alpha = \beta''\cdot c''\cdot\alpha''$. By Def. 7(C),

$$t[\beta\cdot c'\cdot\alpha/x][\tau''\leftarrow c''\cdot\alpha''] \rhd_{I_S} c'\cdot\alpha \in P_{I_S}.$$

Since $t[\beta\cdot c'\cdot\alpha/x][\tau''\leftarrow c''\cdot\alpha''] = t'[c''\cdot\alpha''/x] = t'[\beta'\cdot c'\cdot\alpha/x]$, P_{I_S} contains $t'[\beta'\cdot c'\cdot\alpha/x] \rhd_{I_S} c'\cdot\alpha$ for any α. □

Finally, the next lemma states that if $t[c/x] \Rightarrow_S^* t'[c''/x]$, then $t[o/x] \Rightarrow_{I_S}^* t'[o''/x]$ for some $o \in \nu_S(c)$ and $o'' \in \nu_S(c'')$.

Lemma 6: Let c, $c' \in C$ and t, $t' \in T_M(\{x\})$. If $t[c/x] \Rightarrow_S^* t'[c''/x]$, then there exists a string β such that the first symbol $\beta\cdot c''$ is c and for any $\alpha'' \in C^*$,

$$t[\beta\cdot c''\cdot\alpha''/x] \Rightarrow_{I_S}^* t'[c''\cdot\alpha''/x].$$

Proof: We use induction on the length of the reduction $t[c/x] \Rightarrow_S^* t'[c''/x]$.
Basis: It is obvious when the length of the reduction is zero.
Induction: Consider the following reduction:

$$t[c/x] \Rightarrow_S^* t_i[c_i/x] \Rightarrow_S t'[c''/x].$$

By the inductive hypothesis, there exists a string β_i such that the first symbol of $\beta_i\cdot c_i$ is c and for any $\alpha_i \in C^*$,

$$t[\beta_i\cdot c_i\cdot\alpha_i/x] \Rightarrow_{I_S}^* t_i[c_i\cdot\alpha_i/x].$$

On the other hand, by Def. 8, there exists a subterm t'' of t_i at τ'' such that $t''[c_i/x] \rhd_S c''$ and $t'[c''/x] = t_i[c_i/x][\tau''\leftarrow c'']$. By Lemmas 2 and 5, there exists β' such that the first symbol of $\beta'\cdot c''$ is c_i and for any $\alpha'' \in C^*$, rule $t''[\beta'\cdot c''\cdot\alpha''/x] \rhd_{I_S} c''\cdot\alpha''$ exists in P_{I_S}. By Def. 7, it follows that $t''[\beta'\cdot c''\cdot\alpha''/x] \Rightarrow_{I_S} c''\cdot\alpha''$. Hence,

$$t_i[\beta'\cdot c''\cdot\alpha''/x] \Rightarrow_{I_S} t'[c''\cdot\alpha''/x].$$

We can choose α_i so that $\beta'\cdot c''\cdot\alpha'' = c_i\cdot\alpha_i$. Therefore, it follows that

$$t[\beta_i\cdot\beta'\cdot c''\cdot\alpha''/x] \Rightarrow_{I_S}^* t'[c''\cdot\alpha''/x].$$

Clearly, $\beta = \beta_i\cdot\beta'$ satisfies the condition of the lemma. □

Theorem 3 immediately follows from Lemma 6.

C Complexity Analysis

The algorithm for deciding the sufficient condition stated in Theorem 2 consists of the following three steps:

(S1) Compute Z_0 from S using the type checking algorithm in [12], where Z_0 is a mapping Z whose domain is restricted to $\{m(\mathbf{c}) \mid m \in M_n, \mathbf{c} \in C^n\}$.

(S2) Compute $P_{S,A,Z}$ from S, A, and Z_0.

(S3) Determine whether there exists a class c such that $\tau \Rightarrow_{S,A,Z}^* c$. If such c exists, then output "τ may be insecure." Otherwise, output "τ is secure."

We analyze the time complexity of the algorithm. For the reader's convenience, we explain the notation introduced in Section 4.3 again. Define the size l_t of a term t as $|R(t)|$, i.e., the number of occurrences of t. Define the description length of Σ_c, denoted $\|\Sigma_c\|$, as the sum of l_t for all $(m(\mathbf{c}), t) \in \Sigma_c$. Also, define the size of S, denoted $\|S\|$, as follows:

$$\|S\| = |C| + |\leq| + |M| + |\Sigma_b| + \|\Sigma_c\|.$$

For readability, we use N to mean $\|S\|$. Let k be the maximum arity of all the methods. The height of t, denoted h_t, is defined as the maximum length of the occurrences in $R(t)$. Let L and H be the maximum size and height of all t in $\{t \mid (m(\mathbf{c}), t) \in \Sigma_c\} \cup \{\tau\}$, respectively. We assume that $L \leq N$, i.e., the size of τ does not exceed the size of S.

First, consider (S1). The time complexity of computing $Z_0(m(\mathbf{c}))$ for all $m \in M_n$ and $\mathbf{c} \in C^n$ is

$$\mathcal{O}(kN|C|^{2k+1}),$$

which is given in [12]. In this type-checking algorithm, Z_0 is implemented by a table, and retrieving an element from Z_0 takes $\mathcal{O}(kN|C|^k)$ time. In (S1), we also reconstruct Z_0 by a more efficient data structure such as binomial heap [4]. The time complexity ρ_{Z_0} of retrieving an element from or inserting an element into Z_0 becomes

$$\rho_{Z_0} = \mathcal{O}(k\log(|M|\cdot|C|^k)) = \mathcal{O}(k^2\log N).$$

Note that $\rho_{Z_0} \neq \mathcal{O}(k\log N)$, since the keys are terms $m(\mathbf{c})$ and a key comparison takes $\mathcal{O}(k)$ time. This reconstruction takes

$$\begin{aligned}&\mathcal{O}(|M|\cdot|C|^k(kN|C|^k + \rho_{Z_0}))\\ &= \mathcal{O}(kN|C|^k(N|C|^k + k\log N))\end{aligned}$$

time. Thus, the complexity of (S1) is

$$\mathcal{O}(kN|C|^k(N|C|^k + k\log N)). \qquad (1)$$

```
T1   Q_ans ← ∅, Q_Δ ← ∅
T2   compute Res(m(c)) for all m(c)
T3   for each m(c) in A
T4       add m(c) to Q_Δ
T5       if m ∈ M_c then
T6           let t be Res(m(c))
T7           add t[c/x] to Q_Δ
T8   while Q_Δ ≠ ∅
T9       Q'_Δ ← ∅
T10      for each (t, t') in
                 Q_ans × Q_Δ ∪ Q_Δ × Q_ans ∪ Q_Δ × Q_Δ
T11          if t' is a subterm of t at r' then
T12              if Z(t') has not been computed then
T13                  compute Z(t') from Z_0
T14              for each c' in Z(t')
T15                  add t[r' ← c'] to Q'_Δ
T16      Q_ans ← Q_ans ∪ Q_Δ, Q_Δ ← Q'_Δ
T17  output Q_ans as Q
```

Fig. 9: Procedure for computing Q.

Next, consider (S2). Define $Q = \{t \mid t \rhd_S c \in P_S\}$. In order to compute P_S, it suffices to compute Q, since the right-hand side of $\rhd_S$ can be computed from the left-hand side and Z.

Fig. 9 shows a procedure for computing Q. Suppose that variables Q_{ans}, Q_Δ, Q'_Δ, and Z are implemented by binomial heaps. Let $\rho_{Q_{\text{ans}}}$ denote the complexity of retrieving an element from or inserting an element into Q_{ans}. Define ρ_{Q_Δ}, $\rho_{Q'_\Delta}$, and ρ_Z in the same way. Then,

$$\rho_{Q_{\text{ans}}} = \rho_{Q_\Delta} = \rho_{Q'_\Delta} = \rho_Z = O(L \log |Q|),$$

where L is for a key comparison.

Before analyzing the procedure in Fig. 9 in detail, we estimate $|Q|$. Since it is difficult to estimate $|Q|$ directly, we introduce a finite set Q_0 of terms which possibly appear in the left-hand side of $\rhd_S$. Formally,

$$Q_0 = \bigcup_{(m(\mathbf{c}),t)\in\Sigma_c} X_{t[\mathbf{c}/\mathbf{x}]},$$

where X_t ($t \in T_M(C)$) is defined as follows:

$$X_c = C,$$
$$X_{m(\mathbf{t})} = C \cup \{m(\mathbf{t}') \mid t'_i \in X_{t_i}\}.$$

Intuitively, X_t is the set of all the terms obtained by replacing arbitrary subterms of t with arbitrary classes. Clearly $Q \subseteq Q_0$.

The size of X_t can be obtained by solving the following (in)equalities:

$$|X_c| = |C|,$$
$$|X_{m(\mathbf{t})}| \le |C| + \prod_i |X_{t_i}|.$$

If $k = 1$, then

$$|X_t| \le (h_t + 1)|C|,$$

and therefore,

$$\begin{aligned} |Q_0| &\le \sum_{(m(\mathbf{c}),t)\in\Sigma_c} |X_{t[\mathbf{c}/\mathbf{x}]}| \\ &\le \sum_{(m(\mathbf{c}),t)\in\Sigma_c} (h_{t[\mathbf{c}/\mathbf{x}]} + 1)|C| \\ &= \sum_{(m(\mathbf{c}),t)\in\Sigma_c} l_{t[\mathbf{c}/\mathbf{x}]}|C| \\ &= \|\Sigma_c\| \cdot |C| \\ &\le N|C|, \end{aligned} \tag{2}$$

since $l_t = h_t + 1$ if $k = 1$. Next, consider the case that $k \ge 2$. For any nonnegative integer h, let K_h denote $k^h + k^{h-1} + \cdots + k^0$. In what follows, we show that

$$|X_t| \le (|C| + 1)^{K_{h_t}}. \tag{3}$$

If $h_t = 0$, then $|X_t| = |C| \le (|C|+1)^{K_0} = |C|+1$. Suppose that Eq. (3) holds for any term t' such that $h_{t'} \le h$ for some $h \ge 0$. Consider a term $t = m(\mathbf{t}')$ such that $h_t = h + 1$. Then,

$$\begin{aligned} |X_t| &\le |C| + \prod_i |X_{t'_i}| \\ &\le |C| + ((|C| + 1)^{K_h})^k \\ &= |C| + (|C| + 1)^{K_{h+1}-1} \\ &\le (|C| + 1)^{K_{h+1}}. \end{aligned}$$

Therefore, Eq. (3) holds and

$$\begin{aligned} |Q_0| &\le \sum_{(m(\mathbf{c}),t)\in\Sigma_c} |X_{t[\mathbf{c}/\mathbf{x}]}| \\ &\le \sum_{(m(\mathbf{c}),t)\in\Sigma_c} (|C| + 1)^{K_{h_{t[\mathbf{c}/\mathbf{x}]}}} \\ &\le \|\Sigma_c\|(|C| + 1)^{K_H} \\ &\le N(|C| + 1)^{K_H} \\ &\le N(|C| + 1)^{k^{H+1}}, \end{aligned} \tag{4}$$

using $K_H \le k^{H+1}$ if $k \ge 2$. After all, from Eqs. (2) and (4), we obtain

$$|Q| \le |Q_0| \le N(|C| + 1)^{k^{H+1}}.$$

Let us analyze the procedure in Fig. 9 in detail. See (T2). A straightforward algorithm can compute *Res* in

$$\mathcal{O}(kN|C|^k) \tag{5}$$

time. Next, see (T3) through (T7). In (T3), $|A| \leq |M| \cdot |C|^k \leq N|C|^k$. If *Res* is implemented by an appropriate data structure, then retrieving an element from *Res* takes

$$\rho_{Res} = \mathcal{O}(k \log(|M| \cdot |C|^k)) = \mathcal{O}(k^2 \log N)$$

time in (T6). In (T7), computing $t[\mathbf{c}/\mathbf{x}]$ takes $\mathcal{O}(L)$ time. Therefore, executing (T3) through (T7) takes

$$\begin{aligned} &\mathcal{O}(|A|(\rho_{Q_\Delta} + \log|M_c| + \rho_{Res} + L + \rho_{Q_\Delta})) \\ &= \mathcal{O}(N|C|^k(L \log|Q| + k^2 \log N)) \\ &= \mathcal{O}(k^{H+1}LN|C|^k \log N), \end{aligned} \tag{6}$$

using $k \leq L$.

See (T8) through (T16). By Q_Δ and Q'_Δ, we avoid selecting a duplicated pair of t and t' in (T10). In other words, (T11) through (T15) are executed at most $|Q|^2$ times, and therefore, (T16) is also executed at most $|Q|^2$ times. Moreover, (T13) is executed at most $|Q|$ times, since the condition of (T12) holds at most $|Q|$ times.

In (T11), Knuth-Morris-Pratt string matching algorithm [4] can check in $\mathcal{O}(L)$ time whether t' is a subterm of t. Constructing $t[r' \leftarrow c]$ in (T15) takes $\mathcal{O}(L)$ time. Computing $Q_{\text{ans}} \cup Q_\Delta$ takes $\mathcal{O}(L \log|Q|)$ time [4]. Therefore, the complexity of (T11) through (T16) except (T13) is

$$\begin{aligned} &\mathcal{O}(|Q|^2(L + \rho_Z + \rho_Z + |C|(L + \rho_{Q'_\Delta})) \\ &\qquad + |Q|^2 \cdot L \log|Q|) \\ &= \mathcal{O}(|Q|^2 \cdot |C| \cdot L \log|Q|) \\ &= \mathcal{O}(k^{H+1}LN^2(|C|+1)^{2k^{H+1}+1} \log N). \end{aligned} \tag{7}$$

On the other hand, in (T13), $Z(t)$ is computed from Z_0 as follows:

$$\begin{aligned} Z(m(\mathbf{c})) &= Z_0(m(\mathbf{c})), \\ Z(m(\mathbf{t})) &= \bigcup_{\mathbf{c} \in Z(t_1) \times \cdots \times Z(t_n)} Z_0(m(\mathbf{c})). \end{aligned}$$

The time complexity of computing $Z(t)$ is

$$\mathcal{O}(\rho_{Z_0} l_t |C|^{k+1}) = \mathcal{O}(k^2 L|C|^{k+1} \log N).$$

The total complexity of (T13) is

$$\begin{aligned} &\mathcal{O}(|Q| \cdot k^2 L|C|^{k+1} \log N) \\ &= \mathcal{O}(k^2 LN(|C|+1)^{k^{H+1}+k+1} \log N). \end{aligned} \tag{8}$$

```
U1   D_ans ← ∅, D_Δ ← {τ}
U2   while D_Δ ≠ ∅
U3       D'_Δ ← ∅
U4       for each (t, t'') in D_Δ × Q
U5           if t'' is a subterm of t at r'' then
U6               for each c'' in Z(t'')
U7                   add t[r'' ← c''] to D'_Δ
U8       D_ans ← D_ans ∪ D_Δ, D_Δ ← D'_Δ
U9   if D_ans contains a class then
U10      output "τ may be insecure"
U11  else
U12      output "τ is secure"
```

Fig. 10: Procedure for determining $\tau \Rightarrow_S^* c$.

Both of Eqs. (5) and (6) are bounded by Eq. (7). Furthermore, Eq. (8) is also bounded by Eq. (7) since $k \leq L \leq N$. Thus, the complexity of (S2) is given by Eq. (7).

Lastly, consider (S3). Let $D = \{t \mid \tau \Rightarrow_S^* t\}$. Then,

$$|D| \leq |X_\tau| \leq |Q_0| = \mathcal{O}(N(|C|+1)^{k^{H+1}}),$$

since $h_\tau \leq H$.

Fig. 10 shows a procedure for determining whether D contains a class. Suppose that D_{ans}, D_Δ, D'_Δ are implemented by binomial heaps. By D_Δ and D'_Δ, we avoid selecting $t \in D$ more than once. Therefore, (U5) through (U7) are executed at most $|D| \cdot |Q|$ times. (U8) is also executed at most $|D| \cdot |Q|$ times. Retrieving an element from or inserting an element into D'_Δ takes

$$\rho_{D'_\Delta} = \mathcal{O}(L \log|D|) = \mathcal{O}(k^{H+1}L \log N)$$

time. Computing $D \cup D_\Delta$ also takes $\mathcal{O}(L \log|D|)$ time. Thus, executing (U2) through (U8) takes

$$\begin{aligned} &\mathcal{O}(|D| \cdot |Q|(L + \rho_Z + |C|(L + \rho_{D'_\Delta})) \\ &\qquad + |D| \cdot |Q| \cdot L \log|D|) \\ &= \mathcal{O}(|D| \cdot |Q| \cdot |C| \cdot L \log|D|) \\ &= \mathcal{O}(k^{H+1}LN^2(|C|+1)^{2k^{H+1}+1} \log N). \end{aligned} \tag{9}$$

(U9) can be checked in $\mathcal{O}(|D|)$ time. Therefore, the time complexity of executing (S3) is given by Eq. (9).

By Eqs. (1), (7), and (9), the time complexity of the algorithm is

$$\mathcal{O}(k^{H+1}LN^2(|C|+1)^{2k^{H+1}+1} \log N).$$

A Logical Formalization for Specifying Authorizations in Object Oriented Databases

Yun Bai and Vijay Varadharajan
School of Computing and Information Technology
University of Western Sydney, Nepean
Kingswood, NSW 2747, Australia

Abstract

Specification and reasoning about authorization in object-oriented databases are becoming increasingly significant. Most of the previous work in this field suffers a lack of formal logic semantics to characterize different types of inheritance properties of authorization policies among complex data objects, especially in object-oriented databases. In this paper, we propose a logic formalization to specify object oriented databases together with associated authorizations. Our formalization has a high level language structure to specify object oriented databases and allows various types of authorizations to be specified. Formal semantics is also provided for our formalization. We also illustrate our approach by going through a sample database example and describing the authorization using our formalization.

1 Introduction

Authorization specification in object oriented databases is being increasingly investigated recently by many researchers [4, 6, 8, 9]. However, most of the work todate suffers from a lack of formal logic semantics to characterize different types of inheritance properties of authorization policies among complex data objects. For instance, it is not clear how to provide a formal semantics of inheritance property of access control rights in the context of object oriented databases. Furthermore, it is also difficult to formally reason about authorizations associated with different objects in databases.

The purpose of this paper is to address this issue from a formal logic point of view. In particular, we propose a logical language that has a clear and declarative semantics to specify the structural features of object oriented databases and authorizations associated with complex data objects in databases. Our formalization characterizes the model-theoretic semantics of object oriented databases and authorizations associated with them. A direct advantage of this approach is that we can formally specify and reason about authorizations on data objects without loosing inheritance and abstraction features of object oriented databases. We first propose a logical language for specifying object oriented databases. This language has a high level syntax and its semantics shares some features of Kifer *et al*'s F-logic [7]. We then extend this language for authorization specification. The semantics of the resulting language is defined in such a way that both the inheritance property in an object oriented database (OODB) and authorization rules among different data objects can be formally justified.

The paper is organized as follows. In section 2, we first propose a formal language $\mathcal{L}$ that can be used to specify object oriented databases. A model-theoretic semantics of $\mathcal{L}$ is also provided in this section. In section 3, we extend $\mathcal{L}$ to language $\mathcal{L}^a$ by combining authorization specification associated with data objects into a database. The semantics of $\mathcal{L}^a$ is also provided. In section 4, we investigate properties of reasoning about authorizations in object oriented databases and illustrate a case study using our formalism. Finally, we discuss some related work and our future work in section 5 and section 6 concludes the paper.

2 Formal Language $\mathcal{L}$ for Object Oriented Databases Specification

2.1 Syntax of $\mathcal{L}$

The *vocabulary* of language $\mathcal{L}$ which is used to specify object oriented database consists of:

1. A finite set of *object variables* $\mathcal{OV} = \{o, o_1, o_2, \cdots\}$ and a finite set of *object constants* $\mathcal{OC} = \{O, O_1, O_2, \cdots\}$. We will simply name $\mathcal{O} = \mathcal{OV} \cup \mathcal{OC}$ as *object set*.

2. A finite set $\mathcal{F}$ of function symbols as *object constructors* or *methods* where each $f \in \mathcal{F}$ takes objects as arguments and maps to an object or a set of objects.

3. Auxiliary symbols $\Rightarrow$ and $\mapsto$.

An *object proposition* is an expression of the form

$$\begin{aligned} O \textbf{ has method } & f_1(\cdots) \Rightarrow \Pi_1, \\ & \cdots, \\ & f_m(\cdots) \Rightarrow \Pi_m, \\ & f_{m+1}(\cdots) \mapsto \Pi_{m+1}, \\ & \cdots, \\ & f_n(\cdots) \mapsto \Pi_n. \end{aligned} \quad (1)$$

In (1) O is an object from $\mathcal{O}$ and $f_1, \cdots, f_m, \cdots, f_n$ are function symbols. Each function symbol f takes objects as arguments and maps to some Π that is an object or a set of objects. In an object proposition, a method with the form $f(\cdots) \Rightarrow \Pi$ indicates that f's arguments represent the types of actual objects that should be taken in any instance of this object proposition, and f returns a set of types of the resulting object/objects. On the other hand, a method with the form $f(\cdots) \mapsto \Pi$ indicates that f takes actual objects as arguments and returns an actual object or a set of objects. For example, the following is an object description about a PhD student:

$$\begin{aligned} \mathit{PhDStd} \textbf{ has method } & \mathit{name} \Rightarrow \mathit{String}, \\ & \mathit{area}(\mathit{Staff}) \Rightarrow \mathit{String}, \\ & \mathit{firstdegree} \mapsto \textit{'Bachelor'}, \end{aligned}$$

where $\mathit{name} \Rightarrow \mathit{String}$ represents that the type of name is a string, and method *area* takes type Staff (i.e. another object, eg. the student's supervisor) as a parameter and returns a type of string to indicate the research field, while $\mathit{firstdegree} \rightarrow \textit{'Bachelor'}$ simply expresses that every PhD student should hold a Bachelor degree. An object proposition is called *ground* if there is no object variable occurrence in it.

An *isa proposition* of $\mathcal{L}$ is an expression of one of the following two forms:

$$O \textbf{ isa member of } C, \quad (2)$$

$$O \textbf{ isa subclass of } C, \quad (3)$$

where O and C are objects from $\mathcal{O}$, i.e., O and C may be object constants or variables. Clearly, isa propositions (2) and (3) explicitly represent the hierarchy relation between two objects. An isa proposition without containing any object variables is called *ground isa proposition.*

We call an object or isa proposition a *data proposition.* A data proposition is called *ground data proposition* if there is no object variable occurrence in it. We usually use notation ϕ to denote a data proposition. We assume that any variable occurrence in a data proposition is universally quantified.

A *constraint proposition* is an expression of the form

$$\phi \textbf{ if } \phi_1, \cdots, \phi_k, \quad (4)$$

where $\phi, \phi_1, \cdots, \phi_k$ are data propositions. A constraint proposition represents some relationship among different data objects. With this kind of proposition, we can represent some useful deductive rules of the domain in our database. A *database proposition* is an object proposition, isa proposition, or constraint proposition.

We can now formally define our object oriented database as follows.

Definition 1 *An* object oriented database Σ *is a triplet* (Γ, Δ, Ω), *where* Γ *is a finite set of ground object propositions,* Δ *is a finite set of ground isa propositions, and* Ω *is a finite set of constraint propositions.*

Example 1 We consider a simplified domain about research people in a computer science department. The structure of such domain is illustrated as the following figure.

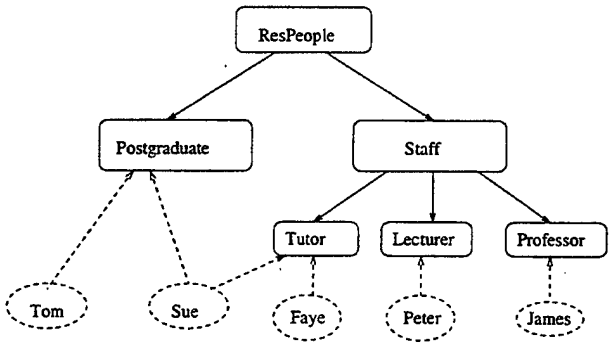


Figure 1: A research people database.

In Figure 1, line arrows indicate subclass relations while dot-line arrows indicate membership relations in the database.

Using our language $\mathcal{L}$, our database $\Sigma = (\Gamma, \Delta, \Omega)$ is specified as follows:
(1) the set of ground object propositions Γ consists of:

$$\begin{aligned} \mathit{ResPeople} \textbf{ has method } & \mathit{name} \Rightarrow \mathit{String}, \\ & \mathit{age} \Rightarrow \mathit{Integer}, \\ & \mathit{firstdegree} \mapsto \textit{'Bachelor'}, \end{aligned} \quad (5)$$

$$\mathit{Postgraduate} \textbf{ has method } \mathit{id} \Rightarrow \mathit{Integer},$$

$$degree \Rightarrow String,$$
$$area(Staff) \Rightarrow String. \quad (6)$$

$$Staff \textbf{ has method} \quad (7)$$
$$research \Rightarrow \{String, \cdots, String\}$$
$$teaching \Rightarrow \{String, \cdots, String\},$$
$$salary \Rightarrow String, \quad (8)$$

$$Tutor \textbf{ has method } topsalary \mapsto \text{'\$45,000'}, \quad (9)$$

$$Lecturer \textbf{ has method } topsalary \mapsto \text{'\$58,000'}, \quad (10)$$

$$professor \textbf{ has method } topsalary \mapsto \text{'\$85,000'}, \quad (11)$$
$$(12)$$

$$Tom \textbf{ has method } name \mapsto \text{'Tom'},$$
$$age \mapsto 21,$$
$$id \mapsto 96007,$$
$$degree \mapsto \text{'Master'},$$
$$area(Peter) \mapsto \text{'Database'}, \quad (13)$$

$$Sue \textbf{ has method } name \mapsto \text{'Sue'},$$
$$age \mapsto 24,$$
$$id \mapsto 95012,$$
$$degree \mapsto \text{'PhD'},$$
$$area(James) \mapsto \text{'Security'},$$
$$salary \mapsto \text{'\$38,000'}, \quad (14)$$

$$Faye \textbf{ has method } name \mapsto \text{'Faye'},$$
$$age \mapsto 30,$$
$$research \mapsto \{\text{'Networking'}\},$$
$$teaching \mapsto \{\text{'CS1'}, \text{'CS2'}\},$$
$$salary \mapsto \text{'\$43,000'}, \quad (15)$$

$$James \textbf{ has method } name \mapsto \text{'James'},$$
$$age \mapsto 42,$$
$$research \mapsto \{\text{'Security'}, \text{'Networking'}\},$$
$$Teaching \mapsto \{\text{'Networking'}\},$$
$$salary \mapsto \text{'\$78,000'}, \quad (16)$$

(2) the set of ground isa propositions Δ consists of:

$$Tom \textbf{ isa member of } Postgraduate, \quad (17)$$
$$Sue \textbf{ isa member of } Postgraduate, \quad (18)$$
$$Sue \textbf{ isa member of } Tutor, \quad (19)$$
$$Faye \textbf{ isa member of } Tutor, \quad (20)$$
$$Ann \textbf{ isa member of } Lecturer, \quad (21)$$
$$James \textbf{ isa member of } Professor, \quad (22)$$
$$Postgraduate \textbf{ isa subclass of } ResPeople, \quad (23)$$
$$Staff \textbf{ isa subclass of } ResPeople, \quad (24)$$
$$Tutor \textbf{ isa subclass of } Staff, \quad (25)$$
$$Lecturer \textbf{ isa subclass of } Staff, \quad (26)$$
$$Professor \textbf{ isa subclass of } Staff, \quad (27)$$

(3) Ω consists of two constraint propositions:

$$y \textbf{ isa member of } Staff$$
$$\textbf{if } x \textbf{ isa member of } Postgraduate,$$
$$x \textbf{ has method } area(y) \mapsto z, \quad (28)$$
$$y \textbf{ has method } research \mapsto \{..., z, ...\}$$
$$\textbf{if } x \textbf{ isa member of } Postgraduate,$$
$$x \textbf{ has method } area(y) \mapsto z, \quad (29)$$

where x, y and z are object variables, and notation $\{..., z, ...\}$ means that set $\{..., z, ...\}$ includes element z while other elements in the set are not interested in here.

In database Σ, we assume that objects *Integer* and *String* be *primitive* object constants that we do not need to give explicit descriptions. That is, we omit isa propositions like 21 **isa member of** *Integer*, *'Tom'* **isa member of** *String* from our database. Our database also presents necessary inheritance properties among different objects. For instance, since object *Tom* is a member of *Postgraduate*, it should inherit method $firstdegree \mapsto$ *'Bachelor'*. The semantics of inheritance will be described in next subsection.

Finally, in Σ, Γ and Δ represent *explicit* data object descriptions and hierarchical relations among these objects, while Ω describes constraints of the domain which characterize some *implicit* data objects an d their properties. By using these rules in Ω and facts in $\Gamma \cup \Delta$, we actually can derive new data objects with some clear properties. For instance, in the above database, we do not give explicit description about object *Peter*. But from proposition (13) and (17) about object *Tom*, we can derive the facts that *Peter* is a member of $Staff$ and has a research field *'Database'*. ■

2.2 Semantics of $\mathcal{L}$

In this subsection, we define the semantics of our database language $\mathcal{L}$ by following a similar way in classical logic.

Definition 2 *Let $\mathcal{L}$ be an object oriented database language we defined earlier. A* structure *of $\mathcal{L}$ is a tuple $I = (U, \mathcal{F}_I, \subseteq_U, \in_U, \Rightarrow_I, \mapsto_I)$, where*

1. *U is a nonempty set called the* universe *of I that representing the set of all* actual *objects in the domain.*

2. *For each n-ary function symbol f in $\mathcal{F}$, there exists a n-ary function f_I: $U^n \rightarrow U$ in $\mathcal{F}_I$. For $n = 0$, f_I is an element of U.*

3. *$\subseteq_U$ is a partial ordering on U and $\in_U$ is a binary relation on U. We require that if $a \in_U b$ and $b \subseteq_U c$, then $a \in_U c$.*

4. *For symbol $\Rightarrow$ in $\mathcal{L}$, $\Rightarrow_I$ is a map: $\Rightarrow_I$: $U \rightarrow (H_0, \cdots, H_i, \cdots)$, where each H_i is a set of $(i+1)$-ary* anti-monotonic *functions with respect to ordering $\subseteq_U$[1] such that for every $h_i \in H_i$,*

$$h_i : U^{i+1} \rightarrow \mathcal{P} \uparrow (U), \tag{30}$$

in which $\mathcal{P} \uparrow (U)$ is the set of all upward-closed *subsets of U with respect to $\subseteq_U$.*

5. *For symbol $\mapsto$ in $\mathcal{L}$, $\mapsto_I$ is a map: $U \rightarrow (G_0, \cdots, G_k, \cdots)$, where G_k is a set of $(k+1)$-ary functions such that for every $g_k \in G_k$,*

$$g_k : U^{k+1} \rightarrow U \cup \mathcal{P}(U), \tag{31}$$

in which $\mathcal{P}(U)$ is the set of all subsets of U.

Now let us look at this definition more closely. In a structure I, U represents all possible *actual* objects in the domain. That is, each element in U is a real object in our world. Then a function symbol (i.e. object constructor) f in $\mathcal{F}$ is corresponding to a function f_I in $\mathcal{F}_I$. Note that f takes object constants or variables in $\mathcal{O}$ as arguments while f_I takes elements in U as arguments and returns an element of U. The function of ordering $\subseteq_U$ is to represent the semantics of isa subclass proposition in $\mathcal{L}$. For example, $a \subseteq_U b$ is the counterpart of proposition A **isa subclass of** B, while A and B are elements in $\mathcal{O}$ (i.e. object constants or variables) and are mapped to a and b, which are the elements of U respectively. We also write $a \subset_U b$ if $a \subseteq_U b$ but $a \neq b$. The semantics of isa membership proposition in $\mathcal{L}$ is provided by $\in_U$ in I in a similar way.

The semantics of $\Rightarrow$, however, is not quite straightforward. As we mentioned earlier, a method with the form $f(\cdots) \Rightarrow \Pi$ actually defines the function type of f. That is, f takes objects that represent types of actual objects and returns an object (or a set of objects) that indicates the type (or types) of resulting actual object (or objects). Suppose f is a i-ary function. Then the semantics of $\Rightarrow$ is provided by mapping $\Rightarrow_I^{(i)}$[2] that maps the resulting object represented by $f(\cdots)$ to a $(i+1)$-ary function $h_i : U^{i+1} \rightarrow \mathcal{P} \uparrow (U)$, where the first ith arguments in U^{i+1} are objects that corresponds to the i arguments taken by f, and the $(i+1)$-th argument in U^{i+1} is the object that corresponds to the object associated with function $f(\cdots)$ in the proposition (we also call the *host object* of f). In $f(\cdots) \Rightarrow \Pi$, Π denotes the type/types of resulting object/objects for which we use a subset of U to represent all the possible actual objects that have type/types indicated by Π.

It is important to note that we require the subset of U to be *upward-closed* with respect to ordering $\subseteq_U$. A subset V of U is upward-closed if for $v \in V$ and $v \subseteq_U v'$, then $v' \in V$. The purpose of this requirement is that if V is viewed as a set of classes, upward closure ensures that for each class $v \in V$, V also contains all the superclasses of v, which will guarantee the proper inheritance property of types.

A similar explanation for $\mapsto_I$ can be given for the semantics of $\mapsto$. We now show that $\Rightarrow_I$ actually provides the type of the corresponding $\mapsto_I$.

To simplify our formalization, we will use *Herbrand universe* in any structures of $\mathcal{L}$. That is, the Herbrand universe U_H is formed from the set of all object constants in $\mathcal{OC}$ and the objects built by function symbols on these object constants.

Definition 3 *Let $I = (U_H, \mathcal{F}_I, \subseteq_I, \in_I, \Rightarrow_I, \mapsto_I)$ be a structure. We define* entailment *relation $\models$ as follows.*

1. *For a ground isa membership proposition, $I \models O$* **isa member of** *C if $O \in_{U_H} C$, and for a isa subclass proposition, $I \models O$* **isa subclass of** *C if $O \subseteq_{U_H} C$*[3].

2. *For a ground object proposition,*

$$I \models O \textbf{ has method } f_1(\cdots) \Rightarrow \Pi_1, \cdots, f_m(\cdots) \Rightarrow \Pi_m, f_{m+1}(\cdots) \mapsto \Pi_{m+1}, \cdots, f_n(\cdots) \mapsto \Pi_n$$

[1] That is, if $u, v \in U^{i+1}$ and $v \subseteq_U u$, then $h_i(v) \supseteq h_i(u)$.

[2] In expression $\Rightarrow_I$: $U \rightarrow (H_0, \cdots, H_i, \cdots)$, we use notation $\Rightarrow_I^{(i)}$ to denote the i-th component of $\Rightarrow_I$, i.e. $\Rightarrow_I^{(i)}$: $U \rightarrow H_i$.

[3] Note that under Herbrand universe U_H, an object constant is mapped to itself in U_H.

if the following conditions hold:

- *for each* $f(O_1, \cdots, O_p)$ *where* f *is in* $\{f_1, \cdots, f_m\}$, $\Rightarrow_I^{(p)} (O')(O_1, \cdots, O_p, O) = \Pi$, *where* $f_I(O_1, \cdots, O_p) = O'$, *and*
- *for each* $f'(O'_1, \cdots, O'_q)$ *where* f *is in* $\{f_{m+1}, \cdots, f_n\}$, $\mapsto_I^{(q)} (O'')(O'_1, \cdots, O'_q, O) = \Pi'$, *where* $f'_I(O'_1, \cdots, O'_q) = O''$.

3. *For a ground constraint proposition,* $I \models \phi$ **if** $\phi_1, \cdots, \phi_k$ *if* $I \models \phi_1, \cdots, I \models \phi_k$ *implies* $I \models \phi$.

4. *For any proposition* ψ *including object variables,* $I \models \psi$ *if for every instance* ϕ *of* ψ *(i.e.* ϕ *is obtained from* ψ *by substituted each variable in* ψ *with some element of* U_H*),* $I \models \phi$.

Now we can formally define the model of a database Σ as follows:

Definition 4 *A structure* M *of* $\mathcal{L}$ *is a* model *of a database* $\Sigma = (\Gamma, \Delta, \Omega)$ *if*

1. *For each proposition* ψ *in* $\Gamma \cup \Delta \cup \Omega$, $M \models \psi$.

2. *For each object proposition* ϕ, *if* $M \models \phi$, *then* $M \models \phi'$ *where* ϕ' *is obtained from* ϕ *by omitting some methods of* ϕ.

3. *For any isa proposition* O **isa member of** C *and object propositions* C **has method** $f(\cdots) \Rightarrow \Pi$ *and* C **has method** $f(\cdots) \mapsto \Pi$, *(1)* $M \models O$ **isa member of** C *and* $M \models C$ **has method** $f(\cdots) \Rightarrow \Pi$ *imply* $M \models O$ **has method** $f(\cdots) \Rightarrow \Pi$; *(2)* $M \models O$ **isa member of** C *and* $M \models C$ **has method** $f(\cdots) \mapsto \Pi$ *imply* $M \models O$ **has method** $f(\cdots) \mapsto \Pi$.

4. *for any isa proposition* O **isa subclass of** C *and object proposition* C **has method** $f(\cdots) \mapsto \Pi$, $M \models O$ **isa subclass of** C *and* $M \models C$ **has method** $f(\cdots) \mapsto \Pi$ *imply* $M \models O$ **has method** $f(\cdots) \mapsto \Pi$.

Condition 1 in the above definition is the basic requirement for a model. Condition 2 allows us to partially represent an object with only those methods that are of interest in a given context. Condition 3 is a restriction to guarantee necessary inheritance of membership, whereas Condition 4 is needed for the purpose of subclass value inheritance.

Let Σ be a database and ϕ be a database proposition. If for every model M of Σ, $M \models \phi$, we also call that ϕ *is entailed* by Σ, denoted as $\Sigma \models \phi$.

3 Databases with Associated Authorizations

In this section, we extend language $\mathcal{L}$ to $\mathcal{L}^a$ to include authorization in object-oriented databases.

3.1 Syntax of $\mathcal{L}^a$

The *vocabulary* of $\mathcal{L}^a$ includes the vocabulary of $\mathcal{L}$ together with the following additions:

1. A finite set of *subject variables* $\mathcal{SV} = \{s, s_1, s_2, \cdots\}$ and a finite set of *subject constants* $\mathcal{SC} = \{S, S_1, S_2, \cdots\}$. We denote $\mathcal{S} = \mathcal{SV} \cup \mathcal{SC}$.

2. A finite set of *access-rights variables* $\mathcal{AV} = \{r, r_1, r_2, \cdots\}$ and a finite set of *access-right constants* $\mathcal{AC} = \{R, R_1, R_2, \cdots\}$. We denote $\mathcal{A} = \mathcal{AV} \cup \mathcal{AC}$.

3. A ternary predicate symbol *holds* taking arguments subject, access-right, and object/method respectively.

4. Logic connectives $\wedge$ and $\neg$.

In language $\mathcal{L}^a$, a fact that a subject S has access right R for object O is represented using a ground atom $holds(S, R, O)$. A fact that S has access right R for object O's method $f(\cdots) \rightsquigarrow \Pi$ is represented by ground atom $holds(S, R, O|f)$. We use $\rightsquigarrow$ for symbol $\Rightarrow$ or $\mapsto$.

In general, we define an *access fact* to be an atomic formula $holds(s, r, o)$ (or $holds(s, r, o|f)$, where $o|f$ indicates a method associated with object o.) or its negation. A ground access fact is an access fact without any variable occurrence. We view $\neg\neg F$ as F. An *access fact expression* in $\mathcal{L}^a$ is defined as follows: (i) each access fact is an access fact expression; (ii) if ψ is an access fact expression and ϕ is an isa or object proposition, then $\psi \wedge \phi$ is an access fact expression; (iii) if ψ and ϕ are access fact expressions, then $\psi \wedge \phi$ is an access fact expression. A ground fact expression is a fact expression with no variable occurrence in it. An access fact expression is *pure* if it does not have an isa proposition occurrence in it.

Based on the above definition, the following are access fact expressions:

$$holds(S, R, O) \wedge O \textbf{isasubclassof} C,$$
$$\neg holds(S, R, o) \wedge o \textbf{isamemberof} C$$

where o is an object variable.

Now we are ready to define propositions in language $\mathcal{L}^a$. Firstly, $\mathcal{L}^a$ has the same types of database propositions as $\mathcal{L}$, i.e. object proposition, isa proposition

and constraint proposition. It also includes the following additional type of *access proposition*:

$$\psi \textbf{ implies } \phi \textbf{ with absence } \gamma, \tag{32}$$

where ψ is an access fact expression, and ϕ and γ are pure access fact expressions. Note that ψ, ϕ and γ may contain variables. In this case, as before, (32) will be treated as a set of access propositions obtained by replacing ψ, ϕ and γ with their ground instances respectively.

As an example, consider the following access proposition:

$$\begin{aligned} &holds(S, Access, Staff|id) \wedge \\ &Alice \textbf{ isa member of } Staff \\ &\textbf{implies } holds(S, Access, Alice|id) \\ &\textbf{with absence } \neg holds(S, Access, Alice|id), \end{aligned}$$

Intuitively, this expression says that if subject S can access staff's id record and Alice is a member of staff, then S can also access Alice's id record if the fact that S cannot access Alice's id record does not currently hold.

There is a special form of access proposition (32) when γ is empty. In this case, we rewrite (32) as

$$\psi \textbf{ provokes } \phi, \tag{33}$$

which is viewed as a *causal* or *conditional* relation between ψ and ϕ. For instance, we may have an access proposition like:

$$\begin{aligned} &holds(s, r, c) \wedge o \textbf{ isa subclass of } c \\ &\textbf{provokes } holds(s, r, o). \end{aligned}$$

This access proposition expresses that for any subject s, access right r and objects o and c, if s has access right r on c and o is a subclass of c, then s also has access right r on o. This is also an example of access inheritance.

On the other hand, there is also a special form of (32) when ψ is empty. In this case, we rewrite (32) simply as

$$\textbf{always } \phi. \tag{34}$$

For example, we can express a fact that the database administrator (DBA) should have any access right on any object as follows:

$$\textbf{always } holds(DBA, r, o).$$

3.2 Databases with Associated Authorizations

It is clear that our access propositions (32), (33) and (34) provide flexibility to express different types of authorization policies on objects. However, to ensure the proper inheritance of access policies on different objects, some specific types of access policies are particularly important for all databases. The set of these kinds of authorization policies is referred to as the *generic authorization scheme* for databases. Consider

$$\begin{aligned} &holds(s, r, o) \textbf{ implies } holds(s, r, o|f) \\ &\textbf{with absence } \neg holds(s, r, o|f). \end{aligned} \tag{35}$$

Intuitively, (35) says that if s has access right r on object o, then s also has access right r on each of its methods under the assumption that $\neg holds(s, r, o|f)$ is not present.

We also have the following two generic access propositions:

$$\begin{aligned} &holds(s, r, c) \wedge o \textbf{ isa subclass of } c \\ &\textbf{implies } holds(s, r, o) \\ &\textbf{with absence } \neg holds(s, r, o), \end{aligned} \tag{36}$$

and

$$\begin{aligned} &holds(s, r, c|f) \wedge o \textbf{ isa subclass of } c \\ &\textbf{implies } holds(s, r, o|f) \\ &\textbf{with absence } \neg holds(s, r, o|f). \end{aligned} \tag{37}$$

(36) and (37) guarantee the proper inheritance of access policies on subclasses.

Finally, the following two propositions ensure the membership inheritance of access policies.

$$\begin{aligned} &holds(s, r, c) \wedge o \textbf{ isa member of } c \\ &\textbf{implies } holds(s, r, o) \\ &\textbf{with absence } \neg holds(s, r, o), \end{aligned} \tag{38}$$

and

$$\begin{aligned} &holds(s, r, c|f) \wedge o \textbf{ isa member of } c \\ &\textbf{implies } holds(s, r, o|f) \\ &\textbf{with absence } \neg holds(s, r, o|f). \end{aligned} \tag{39}$$

Now we can formally define our database with associated authorizations as follows. We will refer to this kind of database as *extended object oriented database.*

Definition 5 *An* extended object oriented database *in* $\mathcal{L}^a$ *is a pair* $\Lambda = (\Sigma, \Xi)$, *where* $\Sigma = (\Gamma, \Delta, \Omega)$ *is the*

database as defined in Definition 1, and $\Xi = GA \cup A$ *is an* authorization description *on* Σ *where GA is a collection of generic authorization propositions (35) - (39), and A is a finite set of user-defined access propositions.*

3.3 Semantics of $\mathcal{L}^a$

Now we consider the semantics of language $\mathcal{L}^a$. To define a proper semantics of our access proposition (32), we need to employ a *fix-point semantics* that shares the spirit of fix-point semantics used for *extended logic programs* [2, 5].

Formally, a *structure* I^Λ of $\mathcal{L}^a$ is a pair (I^Σ, I^Ξ), where I^Σ is a structure of $\mathcal{L}$ as defined in Definition 2 and I^Ξ is a finite set of ground literals with forms $holds(S, R, O)$, $holds(S, R, O|f)$, $\neg holds(S, R, O)$ or $\neg holds(S, R, O|f)$. Now we can define the entailment relation $\models_\lambda$ of $\mathcal{L}^a$.

Definition 6 *Let* $I^\Lambda = (I^\Sigma, I^\Xi)$ *be a structure of* $\mathcal{L}^a$. *We define the* entailment relation $\models_\lambda$ *of* $\mathcal{L}^a$ *as follows.*

1. *For a database proposition* ψ, $I^\lambda \models_\lambda \psi$ *iff* $I^\Sigma \models \psi$.
2. *For a pure ground access fact expression* $\psi \equiv F_1 \wedge \cdots \wedge F_k$, *where each* F_i *is a ground access fact,* $I^\Lambda \models_\lambda \psi$ *iff for each* i, $F_i \in I^\Xi$.
3. *For a ground access fact expression* ψ, $I^\Lambda \models_\lambda \psi$ *iff for each isa or object proposition* ϕ *occurring in* ψ, $I^\Sigma \models \phi$, *and for each ground access fact* ϕ' *occurring in* ψ, $\phi' \in I^\Xi$.
4. *For an access fact expression* ψ, $I^\Lambda \models_\lambda \psi$ *iff for each ground instance* ψ' *of* ψ, $I^\Lambda \models_\lambda \psi'$.

Now we are in the position to formally define a model of $\Lambda = (\Sigma, \Xi)$.

Definition 7 *Consider an extended database* $\Lambda = (\Sigma, \Xi)$ *and a structure* $I^\Lambda = (I^\Sigma, I^\Xi)$. *Let* Ξ' *be an authorization description obtained from* Ξ *in the following way:*

(i) by deleting each access proposition ψ **implies** ϕ **with absence** γ *from* Ξ *if for some* F_i *in* γ, $F_i \in I^{\Xi}$[4];

(ii) by translating all other access propositions ψ **implies** ϕ **with absence** γ *to the form* ψ **provokes** ϕ, *or to the form* **always** ϕ *if* ψ *is empty.*

[4] Recall that $\gamma \equiv F_1 \wedge \cdots \wedge F_k$ is a pure access fact expression, i.e. each F_i $(1 \leq i \leq k)$ is a ground access fact.

Definition 8 *Consider an extended database* $\Lambda = (\Sigma, \Xi)$ *and a structure* $I^\Lambda = (I^\Sigma, I^\Xi)$. *Let* Ξ' *be an authorization description obtained from* Ξ *as described in Definition 5.* $I^\Lambda = (I^\Sigma, I^\Xi)$ *is a* model *of* $\Lambda = (\Sigma, \Xi)$ *if and only if*

(i) I^Σ *is a model of* Σ;

(ii) I^Ξ *is a smallest set satisfying the following conditions:*

(a) for each access proposition **always** ϕ *in* Ξ', $I^\Lambda \models \phi$;

(b) for each access proposition of the form ψ **provokes** ϕ *in* Ξ', *if* $I^\Lambda \models \psi$, *then* $I^\Lambda \models \phi$.

4 Reasoning about Authorizations

In this section, we investigate some properties on the inheritance of authorizations on objects in database. Due to space limit, we cannot provide comprehensive definitions, explanation and examples in here. We just give some theorems to conclude the properties on the inheritance of authorizations. For the proofs of the theorems, refer to the full paper [1].

An extended database may have more than one model. In this case, every model actually represents one possible interpretation for the database with associated authorizations. However, a class of extended databases having unique models presents some interesting properties of authorizations with respect to subclass and membership relationships among objects in databases. An extended database is *well-specified* if it has a unique model.

Theorem 1 *(***Subclass Authorization***) Let* Λ *be a well-specified extended database and S, R, C and O are arbitrary subject constant, access right constant and object constants respectively. Then the following results hold.*

(i) If $\Lambda \models_\lambda holds(S, R, C) \wedge O$ **isa subclass of** C *and* $\Lambda \not\models_\lambda \neg holds(S, R, O)$, *then* $\Lambda \models_\lambda holds(S, R, O)$.

(ii) If $\Lambda \models_\lambda holds(S, R, C|f) \wedge O$ **isa subclass of** C *and* $\Lambda \not\models_\lambda \neg holds(S, R, O|f)$, *then* $\Lambda \models_\lambda holds(S, R, O|f)$.

Theorem 2 *(***Membership Authorization***) Let* Λ *be a well-specified complex database, and S, R, C and O are arbitrary subject constant, access right constant and object constants respectively. Then the following results hold.*

(i) If $\Lambda \models_\lambda holds(S,R,C) \wedge O$ **isa member of** C *and* $\Lambda \not\models_\lambda \neg holds(S,R,O)$, *then* $\Lambda \models_\lambda holds(S,R,O)$.

(ii) If $\Lambda \models_\lambda holds(S,R,C|f) \wedge O$ **isa member of** C *and* $\Lambda \not\models_\lambda \neg holds(S,R,O|f)$, *then* $\Lambda \models_\lambda holds(S,R,O|f)$.

The above two theorems directly follow from generic authorization scheme (35) - (39). The following two theorems, on the other hand, represent that these subclass and membership authorization can be overridden such that the consistency of authorizations can be maintained.

Theorem 3 *(***Overriding of Subclass Authorization***) Let* Λ *be a well-specified complex database, and* S, R, C *and* O *are arbitrary subject constant, access right constant and object constants respectively. Then the following results hold.*

(i) If $\Lambda \models_\lambda holds(S,R,C) \wedge O$ **isa subclass of** C *and* $\Lambda \not\models_\lambda holds(S,R,O)$, *then* $\Lambda \models_\lambda \neg holds(S,R,O)$.

(ii) If $\Lambda \models_\lambda holds(S,R,C|f) \wedge O$ **isa subclass of** C *and* $\Lambda \not\models_\lambda holds(S,R,O|f)$, *then* $\Lambda \models_\lambda \neg holds(S,R,O|f)$.

Theorem 4 *(***Overriding of Membership Authorization***) Let* Λ *be a well-specified complex database, and* S, R, C *and* O *are arbitrary subject constant, access right constant and object constants respectively. Then the following results hold.*

(i) If $\Lambda \models_\lambda holds(S,R,C) \wedge O$ **isa member of** C *and* $\Lambda \not\models_\lambda holds(S,R,O)$, *then* $\Lambda \models_\lambda \neg holds(S,R,O)$.

(ii) If $\Lambda \models_\lambda holds(S,R,C|f) \wedge O$ **isa member of** C *and* $\Lambda \not\models_\lambda holds(S,R,O|f)$, *then* $\Lambda \models_\lambda \neg holds(S,R,O|f)$.

Now, we revisit the research people database domain discussed in Example 1. We will consider a set of authorizations on objects in this database and show how our approach can be used to reason about authorizations in this object oriented database environment.

Example 2 Research people database domain revisited. Consider an extended research people database $\Lambda = (\Sigma, \Xi)$, where Σ is specified to be the database described in Example 1, and Ξ is a set of authorization policies on Σ.

We assume that in this domain, there are three layers of subjects:

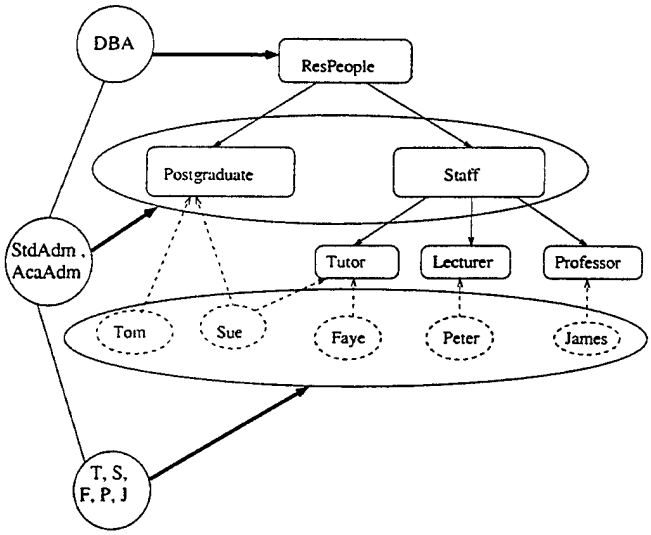


Figure 2: A complex research people database.

(i) A super user called a database administrator DBA, who can read and update class $ResPeople$ and all its subclasses and members through inheritance;

(ii) A student administrator $StdAdm$ who can read and update class $Postgraduate$ and all its members through the inheritance. An academic administrator $AcaAdm$ who can read and update class $Staff$ and all its subclasses and members through inheritance.

(iii) Five individual users named T, S, F, P and J indicating people Tom, Sue, Faye, Peter and James respectively, who can read their own objects $Tom, Sue, Faye, Peter$ and $James$.

The structure of this extended database is shown in Figure 2.

Now we can specify Ξ as follows. Let $\Xi = GA \cup A$, where GA is a collection of (35) - (39) (i.e. generic authorization scheme). Based on the above descriptions (i), (ii) and (iii), A should include the following access propositions:

always $holds(DBA, Read, ResPeople)$, (40)

always $holds(DBA, Update, ResPeople)$, (41)

always $holds(StdAdm, Read, Postgraduate)$, (42)

always $holds(StdAdm, Update, Postgraduate)$, (43)

always $holds(AcaAdm, Read, Staff)$, (44)

always $holds(AcaAdm, Update, Staff)$, (45)

always $holds(T, Read, Tom)$, (46)

always $holds(S, Read, Sue)$, (47)

always $holds(F, Read, Faye)$, (48)

always $holds(P, Read, Peter)$, (49)

always $holds(J, Read, James)$. (50)

(iv) Observing our database structure, it is also clear that *Sue* belongs to two different classes *Postgraduate* and *Tutor*. Due to the membership authorization inheritance property, it turns out that subject *StdAdm* can read and update all methods of *Sue* that are inherited from classes *Staff* and *Tutor*. Clearly, this is not our expectation since intuitively, a student administrator is not usually allowed to access some staff's information (e.g. salary). This problem can be avoided easily by including the following proposition in A:

o **isa member of** *Staff* **implies**
$\neg holds(StdAdm, r, o|salary)$. (51)

(v) Furthermore, if some staff is the supervisor of a postgraduate student, this staff should be able to read all information about his/her student unless there is an explicit declaration denying this. So A also includes one more access proposition:

$holds(s, read, o') \wedge$ (52)
o' **isa member of** *Staff* $\wedge$
o **isa member of** *Postgraduate* $\wedge$
o **has method** $area(o') \mapsto \Pi$
implies $holds(s, Read, o)$
with absence $\neg holds(s, Read, o)$. (53)

Now we have a complete specification for this extended research people object oriented database. It can be proved that this database is well-specified. That is, Λ has a unique model. The following results show that Λ indeed presents the desired authorization policies on objects in the database.

(a) *DBA* can read and update every postgraduate student and staff members' objects and all their methods inherited from all of their superclasses, i.e.
$\Lambda \models_\lambda holds(DBA, Read/Update, O)$[5], and
$\Lambda \models_\lambda holds(DBA, Read/Update, O|f)$,
where O is any object constant in Λ and f is any method associated with O.

[5]It denotes $holds(DBA, Read, O)$ and $holds(DBA, Update, O)$.

(b) *StdAdm* can read and update every student member object, i.e.
$\Lambda \models_\lambda holds(StdAdm, Read/Update, O)$,
where O is *Tom* or *Sue*, which also implies that *StdAdm* can also read and update all methods of *Tom* and *Sue* inherited from their superclasses *ResPeople* and *Postgraduate*, i.e.
$\Lambda \models_\lambda holds(StdAdm, Read/Update, O|f)$.
But *StdAdm* cannot read and update *Sue*'s salary which is inherited from class *Staff*:
$\Lambda \models_\lambda \neg holds(StdAdm, Read/Update, Sue|salary)$.

(c) *AcaAdm* can read and update staff members *Faye* and *James*, i.e.
$\Lambda \models_\lambda holds(AcaAdm, Read/Update, Faye)$, and
$\Lambda \models_\lambda holds(AcaAdm, Read/Update, James)$,
which also imply that *AcaAdm* can also read and update all methods of *Faye* and *James*. Note that since there is no explicit description about object *Peter*, *AcaAdm* cannot access every method of *Peter*.

(d) For every individual user (i.e. T, S, F, P and J), he/she can read his/her corresponding object and all its methods inherited from all its superclasses, i.e.
$\Lambda \models_\lambda holds(T, Read, Tom)$,
$\Lambda \models_\lambda holds(T, Read, Tom|f)$,
$\cdots$,
$\Lambda \models_\lambda holds(J, Read, James)$,
$\Lambda \models_\lambda holds(J, Read, James|f)$,
where f denotes any of the method associated with the corresponding object.

(e) Every supervisor can read his/her student object's all methods, i.e.
$\Lambda \models_\lambda holds(P, Read, Tom|f)$ and
$\Lambda \models_\lambda holds(J, Read, Sue|f')$,
where f and f' denotes any method of *Tom* and *Sue* respectively. Note that $\Lambda \models_\lambda holds(P, Read, Tom|f)$ is derived from the fact that
$\Sigma \models Peter$ **isa member of** *Staff*.

■

5 Discussions and Future Work

Here, we briefly review some related work, discuss the different approaches used in object-oriented database security, and outline our future work.

In [3], a security model for object-oriented databases was proposed. This model consists of a set

of policies, a structure for authorization rules and an algorithm to evaluate access requests. The database is composed of objects that include a collection of facts and a collection of relevant rules. An object binds knowledge rules to database facts. The database is specified by the OSAM* [10, 11] model, in which the generic properties are defined through a *generalization association* and the set of attributes of a class is defined by an *aggregation association.* Derived classes(subclasses) are viewed as generic. Class inheritance properties suggest that access to some attributes of a class also implies access to the corresponding values in its subclass. Generally, there are three types of access policies:

1. A user who has access to a class is allowed to have similar type of access in the corresponding subclasses to the attributes inherited from that class.

2. Access to a complete class implies access to the attributes defined in that class as well as to attributes inherited from a higher class.

3. An attribute defined for a subclass is not accessible by accessing any of its superclass.

In our model, we have considered similar access policies via subclass and membership relationship authorization rules. These rules are specified by the theorems of subclass authorization, membership authorization, overriding subclass authorization and overriding membership authorization. In particular, our **Theorem 1** and **Theorem 3** capture the above three types of access policies. Our model is based on our previous work of formal specification for authorization policies and their transformations, and is discussed from authorization specification point of view. However, in practice, the access policies are organizational dependent. They can vary from organization to organization and can also vary depending on the type of applications.

The access policies that we have considered in this paper are based on subject authorization viewpoint. In practice, both subject and object can be in a hierarchy structure. From object-oriented system viewpoint, access propagation is also data structure related. In our model, authorization propagation from object viewpoint can also be specified. For instance, there is an object *folder*, the *folder* contains a number of *documents*, each *document* contains a number of *files*. We can also specify authorization propagations based on the hierarchical structure. For example, the *documents* can be viewed as subclasses of class *folder*. If the *folder* can be accessed, then the *document* could also be accessed. This can be specified as:

$$holds(s, acess, Folder) \textbf{ implies } holds(s, access, Document).$$

In our future work, we intend to consider attributes for other types of associations. In particular, we will consider in more detail the generalization and aggregation associations. In addition, the placement of the authorization policies also needs to be addressed. They may be placed in a special class or a class they refer to, or propagated through the hierarchy structure. Furthermore, we have not investigated the access to the authorization system itself yet. This will also be considered in our future work.

6 Conclusions

In this paper, we have proposed a logical formalization for specifying authorizations in object oriented databases. Our work consisted of two steps: the first step involved a formal language $\mathcal{L}$ to formalize object oriented databases. We provided a high level language to specify an object oriented database and defined a precise semantics for it. Our semantics of $\mathcal{L}$ shares some features of Kifer *et al*'s F-logic for specifying object oriented databases. But our database specification is more succinct and intuitive, and hence it has been possible to extend this by combining it with the authorization structures. The second step extends $\mathcal{L}$ to language $\mathcal{L}^a$ by representing different types of authorizations in the database. It has been shown that the types of authorizations in our formalism are quite flexible and can be used to reason about complex authorizations compared with other approaches.

References

[1] Y. Bai and V. Varadharajan, A Logical Formalization for Specifying Authorizations in Object Oriented Databases. Manuscript, Nov. 1998.

[2] Y. Bai and V. Varadharajan, A Language for Specifying Sequences of Authorization Transformations and Its Applications. In the *Proceedings of the 1997 International Conference on Information and Communication Security. Springer-Verlag's Lecture Note in Computer Science*, pp39–49, vol 1334, 1997.

[3] E.B. Fernandez, E. Gudes and H. Song, A Security Model For Object-Oriented Databases. *Proceedings of IEEE Symposium on Research in Security and Privacy*, pp 110–115, 1989.

[4] E.B. Fernandez, R.B. France and D. Wei, A Formal Specification of An Authorization model for Object-Oriented Databases. In *Database Security, IX: Status and Prospects*, Elsevier Science Publishers B. V., pp 95-109, 1995.

[5] M. Gelfond and V. Lifschitz, Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 9:365-385, 1991.

[6] E. Gudes, H. Song, E.B. Fernandez, Evaluation of Negative, Predicate, and Instance-Based Authorization in Object-Oriented Databases. In *Database Security, IV: Status and Prospects*, S. Jajodia and C.E. Landwehr (Editors). Elsevier Science Publishers B. V., pp 85-98, 1991.

[7] M. Kifer, G. Lausen, J. Wu, Logical Foundations of Object-Oriented and Frame-Based Languages. *Journal of ACM*, vol. 42, No. 4 (July), pp 741-843, 1995.

[8] T.F. Lunt, Discretionary Security for Object-Oriented Database Systems. Technical Report 7543, Computer Science Laboratory, SRI International, 1990.

[9] J.K. Millen, T.F. Lunt, Security for Object-Oriented Database Systems. *Proceedings of IEEE Symposium on Research in Security and Privacy*, pp 260–272, 1992.

[10] S.Y.W. Su and L. Raschid, Incorporating Knowledge Rules in a Semantic Data Model: An Approach to Integrated Knowledge Management, *Proceedings of AI Applications Conference*, 1985.

[11] S.Y.W. Su, V. Krishnamurthy and H. Lam, An Objected-Oriented Semantic Association Model (OSAM*), *AI in Industrial Engineering and Manufacturing: Theoretical Issues and Applications*, S. Kumara, R. Kashyap and A.L. Soyster (Editors). ALLE, 1998.